(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－27121

(43)公開日　平成10年(1998)１月27日

(51)Int.Cl.⁶		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	12/00	５３３		Ｇ０６Ｆ	12/00	５３３Ｊ
		５３１				５３１Ｒ
		５３５				５３５Ｚ
	17/30				15/40	３１０Ｆ
					15/401	３４０Ａ

審査請求　未請求　請求項の数８　ＯＬ　外国語出願　（全 99 頁）

(21)出願番号　特願平9－7167

(22)出願日　平成９年(1997)１月20日

(31)優先権主張番号　５８８１４２

(32)優先日　1996年１月18日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　597004720
サン・マイクロシステムズ・インコーポレーテッド
Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．
アメリカ合衆国カリフォルニア州94043－1100，マウンテン・ヴュー，ガーシア・アヴェニュー　2550，エムエス・ピーエイエル１－521

(74)代理人　弁理士　社本　一夫　（外４名）

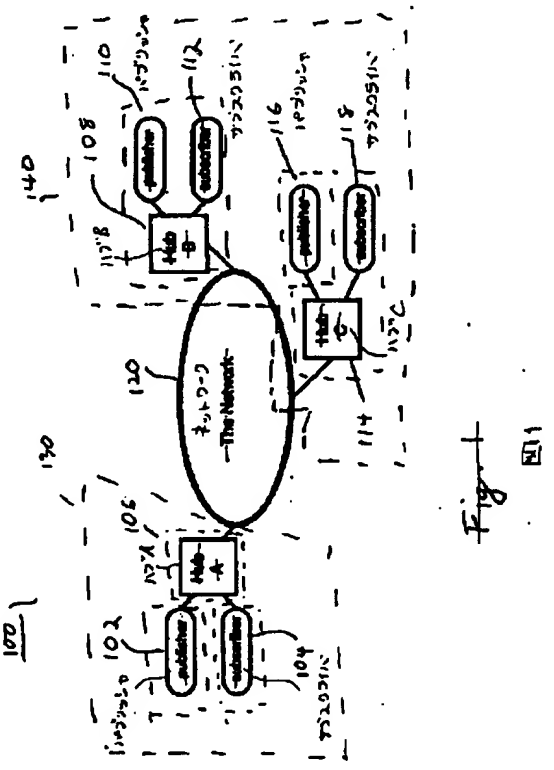
最終頁に続く

(54)【発明の名称】　データベース・ネットワークの接続性に関する生産物

(57)【要約】　（修正有）

【課題】　サポートが困難なレガシ・データベース・ソフトウェアを、修正を加えずにネットワーク・システムに統合できるようにする。

【解決手段】　「公表と予約」モデルを採用し、レガシ・データベースのデータベース接続部を含むパブリシャ１０２～１１６やサブスクライバ１０４～１１８がシステム内の通信の基礎となる「イベント」を公表あるいは予約できるようにする。パブリシャとサブスクライバは互いについての直接の知識をもたず、「広告」を行うことにより公表と予約を非同期的に行う。サブスクライバは公表されたイベントであって指定した予約基準と一致するものを受信できる。データベース接続部はレガシ・データベースのトランザクション・モニタに対してはノードとして振る舞い、ネットワークのハブ１０６～１１４に対してはクライアントとして公表や予約を行う。



【特許請求の範囲】
【請求項１】データ・ベースをパブリッシャ／サブスクライバ・ネットワークに接続することにより、前記データ・ベースがイベントをネットワークに公表可能とする方法であって、
トランザクション・モニタ・コンピュータ・システムを通じて、前記データ・ベースがデータ・ベース接続コンピュータ・プログラムと通信可能とするリンクを用意するステップと、
データ・ベース接続部を、前記ネットワーク内のパブリッシャ・ハブとして設定するステップと、
前記データ・ベースにより、入力を前記データ・ベース内のデータを変更するユーザから受け取るステップと、
前記トランザクション・モニタに、前記データが変更されたことを通知するステップと、
前記変更データを受け入れるべきことを示す入力をユーザから受信するステップと、
前記トランザクション・モニタに前記変更データを受け入れることを通知するステップと、
前記データ・ベース接続部によって、前記データを受け入れたという前記トランザクション・モニタからのメッセージにしたがって、前記変更データにしたがって、前記ネットワークに、公表されたイベントを送出するステップと、から成り、前記ステップはデータ処理システムによって実行されることを特徴とする方法。
【請求項２】請求項１記載の方法であって、更に、
前記変更データをロール・バックすべきことを指示する、ユーザからの入力を受け取るステップと、
前記トランザクション・モニタに、前記変更データをロール・バックすることを通知するステップと、
前記データ・ベース接続部によって、前記ロールバックにしたがって、前記変更データを前記ネットワークに送ることを自制するステップと、を含むことを特徴とする方法。
【請求項３】データ・ベースをパブリッシャ／サブスクライバ・ネットワークに接続することにより、前記データ・ベースがネットワーク上のイベントに予約できるようにする方法であって、
データ・ベース接続コンピュータ・プログラムが前記データ・ベースと通信可能とするリンクを用意するステップと、
前記データ・ベース接続部を、前記ネットワークにおけるサブスクライバ・ハブとして設定するステップと、
前記データ・ベース接続部によって、前記ネットワークからイベントを受信するステップと、
前記データ・ベース接続部によって、前記受信したイベントに応じたデータを前記データ・ベースに送出するステップと、
前記データ・ベース接続部によって、前記データ・ベース内に前記データを受け入れるステップと、から成り、前記ステップはデータ処理システムによって実行されることを特徴とする方法。
【請求項４】請求項１記載の方法であって、更に、
前記データ・ベース接続部によって、前記イベントをその近隣ハブの１つに送出するステップを含み、前記データ・ベース接続部内のデータ構造は、前記近隣ハブが、前記イベントに対するサブスクライバへの最少コスト上にあることを示すことを特徴とする方法。
【請求項５】請求項１記載の方法であって、更に、
前記データ・ベース接続部によってイベントを受信するステップと、
前記データ・ベース接続部によって、前記データ・ベース接続部のデータ構造にしたがって、前記データ・ベースが前記イベントに予約されていることを判定するステップと、
前記データ・ベース接続部によって、前記イベントを前記データ・ベースに送出するステップと、を含むことを特徴とする方法。
【請求項６】請求項１記載の方法であって、更に、
前記データ・ベース接続部によってイベントを受信するステップと、
前記データ・ベース接続部によって、前記データ・ベース接続部のデータ構造にしたがって、近隣ハブが、前記イベントに対するサブスクライバへの最少コスト経路上に直接的または間接的に接続されていることを判定するステップと、
前記データ・ベース接続部によって、前記イベントをその近隣ハブに送出し、前記イベントを前記サブスクライバに転送可能とするステップと、を含むことを特徴とする方法。
【請求項７】請求項１記載の方法において、前記送出ステップは、
前記データ・ベース接続部内のデータ構造が、前記データベースがイベントのタイプを有するイベントに予約することを示すとき、前記データ・ベース接続部によって、前記近隣ハブの１つに前記イベントを送出するステップを含むことを特徴とする方法。
【請求項８】請求項１記載の方法において、前記送出ステップは、
前記データ・ベース接続部のデータ構造内の内容フィルタが、前記データ・ベースがイベントの中で発見した値を有するイベントに予約することを示すとき、前記データ・ベース接続部によって、その近隣ハブの１つに前記イベントを送出するステップを含むことを特徴とする方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
関連出願
"Middleware For Enterprise Information Distributio

n"と題する、BrachoおよびJankowskiの米国特許出願番号第　　号が本願と同時に出願された。

【0002】追加資料

追加資料Aは、本明細書の一部であり、この言及により本願にも含まれているものとする。この追加資料Aには、本発明の好適実施例において、ハブ間で渡すことができるシステム・イベントの一例について、その概要が収容されている。

【0003】本発明は、情報公表システムに関し、更に特定すれば、レガシ・データ・ベース(legacy data base)を情報公表システムに接続する方法および装置に関するものである。

【0004】

【従来の技術】従来のシステムの中には、データ処理システムのノード間で情報を交換するために「トランザクション」モデル(transactional model)を使用するものがある。

【0005】従来のトランザクション・モデルでは、2つのアプリケーションが同期した「トランザクション」を起動し、このトランザクションは、ノード間で情報を交換している限り保持される。トランザクション・モデルは、各業務アクティビティ(business activity)にトランザクションを定義することを必要とする。ネットワークの広範囲にわたる異なる部分にあるノードがトランザクション・モデルを通じて通信しようとする場合、当該システムの全ての部分で良好に動作するトランザクションを定義することが困難なことが多い。更に、トランザクションは、当該トランザクションを開始したノードのみを伴うことができる。他のノードは途中からトランザクションに予約することはできない。この方式は、ノードがシステム内の他の全ノードについて知らないようなシステムには、不向きである。

【0006】多くの商業的企業は未だに、昔開発されもはやその製造者によるサポートを受けられないソフトウエアを使用している。このようなソフトウエアのことを「レガシ・ソフトウエア」と呼ぶ。加えて、多くの商業的企業は、商用ソフトウエア製品を使用する。既存の商用ソフトウエアやレガシ・ソフトウエアを修正し、新たにネットワーク組織としたシステムで動作させることは、必ずしも可能ではなく、また望ましいことでもない。商用ソフトウエアおよびレガシ・ソフトウエアは非常に複雑な場合があり、それを修正するのは困難で、しかも費用がかかる。会社は、常に稼働している安定なシステムに修正を加えることには、慎重である。更に、会社内には、修正を加えることができる程にレガシ・システムについて詳しく理解しているコンピュータ・プログラマが全くいない場合もある。最後に、会社は、レガシ・アプリケーションを商業的販売人から購入した場合、当該レガシ・アプリケーションのソース・コードを所有していないという場合もある。また、会社は、データ・ベース・ソフトウエアを修正する法的な権利を有していない場合もある。

【0007】

【発明が解決しようとする課題】必要とされているのは、商用およびレガシ・データ・ベース・ソフトウエアをネットワーク・システムに統合し、しかもデータ・ベース・ソフトウエア自体には修正を加えなくて済む方法である。

【0008】

【課題を解決するための手段】本発明は、従来技術の問題点および欠点を克服するにあたり、ネットワーク内でパブリッシャあるいはサブスクライバのいずれとしても動作可能な「データ・ベース接続部」を提供する。

【0009】本発明は、「ミドルウエア」(middleware)の一種として実施される。ミドルウエアとは、アプリケーション・プログラムと制御レベルのプログラムとの間に位置するソフトウエアのことである。ミドルウエアは「ネットワーク中心」(network centric)であり、特定のユーザ・インターフェースや特定のデータベースの組織に集中しない。以下に記載する実施例は、情報を公表する複数の「パブリッシャ」(publisher)エンティティと、情報を要求し使用する複数の「サブスクライバ」(subscriber)エンティティとを含む。パブリッシャとサブスクライバは、互いにネットワークで接続されている。ネットワークは、「格納および転送」(store and forward)ネットワークであり、そのルーティング(routing)は「内容を基本とする」(content-based)ものである。内容を基本とするルーティング・システムでは、情報は、当該情報の内容に基づいて流通するのであり、システム内のパブリッシャやサブスクライバのアドレスに基づくのではない。以下に記載する実施例では、情報は多くのサブスクライバに並行して分配される。

【0010】以下に記載する実施例では、情報の基本的単位(basic quanta)を「イベント」(event)と呼ぶ。パブリッシャはイベントを公表し、サブスクライバは、サブスクライバ自身が定義した基準と一致するイベントに予約を行う。以下に記載する実施例では、イベントを表わす際に、Cプログラム言語におけるデータ構造を使用するが、適切な表現であればあらゆるものが使用可能である。

【0011】公表と予約は非同期的に実行される。パブリッシャとサブスクライバは互いの直接的な知識を有していない。システムは、パブリッシャから公表されたイベントを受信し、全ての適切なサブスクライバにそのイベントを送出する。各サブスクライバは、イベントがサブスクライバ自身の指定した予約基準に一致する場合のみ、システム上で公表された全てのイベントを受信することを保証されている。

【0012】以下に記載する実施例は、パブリッシャおよびサブスクライバ用のアプリケーション・プログラミ

ング・インターフェース（ＡＰＩ:Application ProgrammingInterface)を含む。ＡＰＩは、各パブリッシャおよびサブスクライバにシステムへのインターフェースを可能にする、複数の手順を定義する。したがって、様々なタイプのパブリッシャおよびサブスクライバは、ＡＰＩにしたがって定義されたインターフェース手順を使用している限り、システムに接続することができる。また、以下に記載する実施例は、共通のデータベース、インストールおよび管理ツール、およびロギング機能(logging facilities)のような、複数のソフトウエア・エレメントに対するサポートを含む。したがって、以下に記載する実施例は、商業用企業の全域にわたって使用可能なので、「企業システム」と命名する。

【００１３】以下に記載する本発明の実施例は、レガシ・システム、レガシ・アプリケーション、およびレガシ・ハードウエアをシステムに容易に統合し、ユーザがシステムを使用するために修得しなければならない情報量を最少に抑えようとするものである。システム内の通信は非同期の「イベント」に基づいて行われるので、本発明は、様々なオペレーティング・システムの下で実行し、多種多様なアプリケーションを走らせる、ＰＣおよびメインフレームの双方を含む異種ネットワーク上での実施が可能である。以下に記載する実施例は、分散オブジェクト環境(distributed-object environment)を利用し、本発明の好適実施例は、共通オブジェクト要求ブローカ(CORBA:Common Object Request Broker)を実施する。

【００１４】本発明の目的によると、本明細書において具体化され広く記載されるように、本発明は、データ・ベースをパブリッシャ／サブスクライバ・ネットワークに接続し、データ・ベースがネットワークにイベントを公表できるようにする方法に関する。この方法は、データ処理システムによって実行される次のステップから成る。即ち、リンクを用意し、トランザクション監視コンピュータ・プログラムによって、データ・ベースがデータ・ベース接続コンピュータ・プログラムと通信できるようにするステップと、ネットワークにおいてデータ・ベース接続部をパブリッシャ・ハブとして設定するステップと、データ・ベースによって、当該データ・ベース内のデータを変更するユーザからの入力を受信するステップと、データが変更されたことをトランザクション・モニタに通知するステップと、変更データを受け入れることを示すユーザからの入力を受信するステップと、変更データが受け入れられたことをトランザクション・モニタに通知するステップと、データ・ベース接続部によって、データが受け入れられたというトランザクション・モニタからのメッセージにしたがって、公表されたイベントを、変更されたデータにしたがってネットワークに送るステップから成る。

【００１５】本発明の目的および利点は、部分的に以下の説明に記載されており、部分的に以下の記載から明白であり、あるいは本発明の実施を通じて学習することができる。本発明の目的および利点は、特許請求の範囲において特定的に指摘してある要素および組み合わせ、ならびにその均等物によって、実現および達成されよう。

【００１６】添付図面は、本明細書に含まれその一部を構成するが、本発明の種々の実施例を例示し、以下の記載と共に、本発明の原理を説明するために供するものである。

【００１７】

【発明の実施の形態】これより、本発明の好適実施例について詳細に説明する。本発明の例は添付図面に示されており、可能な場合は、同一参照番号を図面全てにおいて使用して、同一部分または同様部分に言及することとする。

【００１８】１．概要

図１は、本発明の好適実施例によるネットワーク型コンピュータ・システム１００のブロック図である。ネットワーク型コンピュータ・システム１００は、複数のパブリッシャ１０２，１１０，１１６および複数のサブスクライバ１０４，１１２，１１８を含む。パブリッシャ１０２とサブスクライバ１０４は、ハブ１０６を通じてネットワーク１２０に接続されている。パブリッシャ１１０およびサブスクライバ１１２はハブ１０８を通じてネットワーク１２０に接続されている。パブリッシャ１１６とサブスクライバ１１８は、ハブ１１４を通じてネットワーク１２０に接続されている。ハブ１０６およびそれに接続されているパブリッシャおよびサブスクライバは第１テリトリ(territory)１３０にある。ハブ１０８，１１４およびこれらに接続されているパブリッシャおよびサブスクライバは第２テリトリ１４０にある。その他のテリトリ（図示せず）もネットワーク１２０には存在する。「ハブ」、「パブリッシャ」、「サブスクライバ」、および「テリトリ」については、以下で順番に論じることにする。

【００１９】図１において点線で示すように、各パブリッシャ、サブスクライバ、およびハブは、別個のコンピュータ（１０２，１０４，１０６）、同一コンピュータ（１０８，１１０，１１２）、または別個のコンピュータおよび同一コンピュータ（１１４，１１６，１１８）のいずれかの組み合わせの中に配置することができる。ハブは、０カ所以上のパブリッシャおよび０カ所以上のサブスクライバを有することができる。また、ハブは直接その他のハブと接続することも可能である。

【００２０】各コンピュータ（図１において点線で表わす）は、ＣＰＵと、メモリと、入出力線とを含むことは、当業者には理解されよう。これらの要素は、例の明確化のために、図には示していない。各コンピュータは、ディスク駆動装置、キーボード、表示装置、ネットワーク接続部、追加メモリ、追加ＣＰＵ等のようなその

他の多数の要素を含むことも可能である。パブリッシャ、サブスクライバ、およびハブは、メモリ内に格納されたソフトウエア命令として実施され、コンピュータ・システムのプロセッサによって実行されることが好ましい。通常、パブリッシャ、サブスクライバ、およびハブ・ソフトウエアは、当該ソフトウエアが格納されているコンピュータのプロセッサによって実行されるが、他のコンピュータ・システムのプロセッサがかかるソフトウエアを実行することも可能である。

【００２１】本発明の好適実施例は、ソラリス(Solaris)オペレーティング・システム、バージョン２．４の下で実行する。ソラリスは、米国およびその他の国においてサン・マイクロシステムズ・インコーポレーテッド(Sun Microsystems, Inc.)の登録商標であり、ユニックス(Ｕｎｉｘ)オペレーティング・システムを基本とするものである。ユニックスは、米国およびその他の国における登録商標であり、Ｘ／ＯＰＥＮ，Ｌｔｄ．を通じて独占的に実施許諾されている。ネットワーク型コンピュータ・システム１００は、ＴＣＰ／ＩＰのようなネットワーク通信プロトコルを使用するが、本発明を実施するには、あらゆる適切なプロトコルを使用することができる。

【００２２】ここに記載する実施例では、「パブリッシャ」はネットワーク１２０上である種のイベントを公表し、「サブスクライバ」はある種のイベントに予約を行う。パブリッシャおよびサブスクライバは非同期的に動作し、互いの存在について知らない。「公表および予約」モデルを使用することにより、ネットワークからにアプリケーションを、および互い分離することができる。公表は、イベントの同報通信として考えることができる。イベント・タイプは無線局と類似している。特定の局に関心があるアプリケーションは、特定のイベント・タイプに予約（即ち、渡す）する。関連する当事者が予め１組の可能な周波数で同意していなければならない、無線の同報通信と丁度同じように、ここに記載する実施例におけるアプリケーションも、所定の組のイベント・タイプに予め同意していなければならない。

【００２３】ここに記載する本発明の実施例は、「内容による予約」を使用する。サブスクライバは、イベント・タイプおよび時報の内容に基づいて、彼らがほしいものを指定する。ここに記載する実施例は、イベントに対して、「格納および転送」技法を利用する。この用語は、パブリッシャがオフ・ラインであても、サブスクライバがイベントを取り込むことができるように、システムがイベントをバッファに入れておくことを意味する。同様に、パブリッシャも、サブスクライバがオフ・ラインであっても、イベントを公表することができる。

【００２４】ここに記載する実施例は、ハブ単位で管理される。ハブ１０６のようなハブは、パブリッシャおよび／またはサブスクライバの局在接続点である。パブリッシャおよびサブスクライバ双方を、ハブの「クライアント」と呼ぶ。パブリッシャは「広告」(advertisement)を用いて、それが公表しようとしているイベントのタイプ、どのようにそれらを公表しようとしているか（例えば、毎日、各イベント・トリガが発生する度、等）について、システムに伝える。広告は、パブリッシャのインストールの間に、このパブリッシャが接続されているハブに登録される。広告は固有の名称を有する必要がある。公表すべきイベント・タイプの名称を含むことに加えて、広告はパブリッシャについての他の情報も含む。この情報は、公表されるイベントの優先度レベル、イベントの有効期間等を含むことができる。ハブは広告を全ての潜在的サブスクライバに送信する。ハブは、公表されたイベントを、サブスクライバの予約と内容が一致するタイプのイベントに予約してある、全てのサブスクライバに送信する。

【００２５】ＩＩ．パブリッシャおよびサブスクライバ

以下の章では、アプリケーション（パブリッシャまたはサブスクライバのいずれか）を作成し、それをハブ上にインストールするために必要なステップについて説明する。また、以下の章では、パブリッシャの一例がイベントを公表する際に実行するステップ、およびサブスクライバの一例がイベントに予約する際に実行するステップについても説明する。ここに記載する実施例は、プログラマが高水準言語でアプリケーション（パブリッシャおよびサブスクライバ）用ソフトウエアを書き、ＡＰＩに定義されているルーチンを使用し、更に各ユーザ定義イベントのタイプに対して発生したイベント操作ルーチンを使用して、ハブとインターフェースを行うことができるようにするものである。

【００２６】図２は、例えば、パブリッシャのようなアプリケーションをハブ上にインストールする際に実行するステップを示すフロー・チャートである。図３は、インストールしたパブリッシャがあるイベントを公表する際に実行するステップを示すフロー・チャートである。図４は、図１のサブスクライバが、あるタイプおよび内容のイベントに予約する際に実行するステップを示すフロー・チャートである。先に論じたように、パブリッシャおよびサブスクライバは双方とも、プロセッサによって実行されるソフトウエア・プログラムとして実施することが好ましい。

【００２７】ここに記載する実施例は、イベントの送信（公表）および受信（予約）を中心とする。パブリッシャがイベントを公表できるようにするには、パブリッシャはそれが公表しようとするイベントを定義し、その広告を行わなければならない。イベントを意味のあるものにするために、パブリッシャおよびサブスクライバは互いに理解し合う必要がある。このために、ここに記載する実施例は、イベントを定義するために、標準的な使用言語を使用する。図２のステップ２０２において、パブ

リッシャは、当該パブリッシャが公表可能な1つ以上のイベントを定義する。ここに記載する実施例は、業界標準のインターフェース定義言語（ＩＤＬ:Interface Definition Language）を用いて、イベントの定義が可能である。ＩＤＬは、オブジェクト管理グループ（ＯＭＧ:Object Management Group）によって普及された標準的インターフェースであるが、イベントの構造を記述するために適用可能ないずれのシンタックスも、ここに記載する実施例と組み合わせて使用可能である。ステップ２０４において、イベントのＯＭＧ ＩＤＬ定義をＣプログラミング言語のデータ構造に変換する。

【００２８】例えば、アプリケーションが"SalesOrder"イベントを公表すると仮定する。このアプリケーションをシステムに追加可能にするには、そして他にSalesOrderイベントを扱うアプリケーションが未だ存在しないと仮定すると、プログラマはＯＭＧＩ ＤＦを用いて、以下のように、"SalesOrder"を定義する。

【００２９】

```
//filename: order:ddl
interface SalesOrder{
        struct Address{
                string street;
                string city;
                string state;
                unsigned long zip;
        };
attribute string customer_name;
attribute Address customer_address;
};
```

【００３０】上記のシンタックスは、１つのイベントを「インターフェース」として定義し、当該イベント内のデータ・メンバを「属性」として定義する。

【００３１】１つ以上のイベントを定義した後、これらのイベントをＣプログラミング言語の構造体定義に変換する。好ましくは、この変換は、ＯＭＧ ＩＤＬコンパイラによって実行し、ヘッダ・ファイルとコード・ファイルを各イベントについて発生する。発生したヘッダ・ファイルは、コード・ファイル内に定義した型宣言された関数(typed function)に対するイベントの宣言を表わす、単純なＣ言語構造体を含む。タイプSalesOrderの上記の例示イベントのために発生したＣ構造体の定義は、以下の通りである。

【００３２】

```
typedef struct SalesOrder_Address {
        nxsString street;
        nxsString city;
        nxsString state;
        nxsULong zip;
}  SalesOrder_Address;

typedef struct SalesOrder {
        nxsStrig customer_name;
        SalesOrder_address customer_address;
} SalesOrder;
```

【００３３】この例のＩＤＬがどのようにして上に示すＣプログラミング言語構造の定義に変換するかについては、当業者は容易に理解しよう。

【００３４】イベント定義から発生したコード・ファイルは、定義されたタイプ（ここでは、SalesOrderイベント）のイベントを操作するための手順を含む。この手順は少なくとも以下の処理を行う。

【００３５】パブリッシャによる使用のため
タイプSalesOrderのイベント作成。タイプSalesOrderのイベント公表。タイプSalesOrderのイベント廃棄。
サブスクライバによる使用のため
タイプSalesOrderのイベント内容の印刷。タイプSalesOrderのイベントに対する予約登録。

【００３６】上記の複数の手順は各定義されたイベントについて発生するが、これら手順の処理は、イベントの構造における相違によって異なるものとなることは、当業者は理解しよう。また、これらの手順は、ここに記載する実施例がプログラミング・エラーを防止するためにタイプされた関数を用いるという事実によっても、異なるものとなる。

【００３７】ハブにプロセス・アプリケーションをインストールするプロセスは、ステップ２０６に示すように、１つ以上の広告をハブに格納することを含む。これらの広告は、ハブが知っているイベントのタイプを定義

する。1つの広告は、当該広告の名称、パブリッシャが公表するイベント（例えば、SalesOrderのイベント）の記述、どのようにパブリッシャがそのイベントを公表するか（例えば、毎日、イベント・トリガが発生したとき等）、イベント・タイプの名称、イベントの優先度レベル、および当該イベントの消滅時間（「存続時間」とも呼ぶ）を含む。図示していないが、ここに記載する実施例は、種々のイベント・タイプについてアクセス制御許可を設定することも可能である。

【００３８】また、ここに記載する実施例は、ＡＰＩにおいて予め定義されたルーチンを含み、これらはパブリッシャおよびサブスクライバ・ソフトウエアには使用可能であり、少なくとも以下の機能を実行するルーチンを含む。

【００３９】クライアントの登録。既存のセッションを用いてのクライアントの再確立。クライアントの抹消。公表する意向の登録。公表の抹消。イベントの登録。イベントへの予約。イベントへのアクティブな予約の再活性化。予約の取り消し。

【００４０】本発明による他のＡＰＩは、多少異なるＡＰＩ機能を含むこともあり得る。加えて、ＡＰＩは、呼び出し元のアプリケーションがあるタイプのイベントの予約または公表が可能であるか否かを示すブール値を返す、"can_subscribe"ルーチン（予約可能ルーチン）および"can_publish"ルーチン（公表可能ルーチン）を含むことが好ましい。また、ＡＰＩは、あるイベントのパブリッシャについての情報、およびどのようにしてそのイベントを公表したかについての情報を返す、"get_current_envelope"ルーチン（現エンベロープ取得ルーチン）も含む。（このルーチンは、コールバック手順からのみコールされる）。例えば、サブスクライバがあるイベントを受信した場合、コールバック手順をコールする。図9は、あるイベントに対する、エンベロープ・データ構造のフォーマットを示す。

【００４１】あるパブリッシャまたはサブスクライバについてのイベント記述を最初に、例えば、Ｃに変換するとき、先に発生について論じたイベント・ヘッダ・ファイルを、アプリケーション・ソフトウエア・ソース・コードに含ませる。したがって、この例では、パブリッシャは、SalesOrderイベントの定義へのアクセスを有する。また、変換はイベント操作ルーチンおよびＡＰＩルーチンを、この時点でアプリケーションにリンクする。対応するハブは、その他のハブと共に、広告の存在を示す通知を、潜在的なサブスクライバに送信する。これについては、以下で論ずる。

【００４２】一旦パブリッシャをハブにインストールしたなら、広告およびイベントを自由に公表することができる。図3は、パブリッシャの処理の間に、図1のパブリッシャが実行するステップを示すフロー・チャートである。ＩＤＬをコンパイルすることによって作成されたイベント操作ルーチンへのコールを伴うステップは、図においてアステリスクによって示されている。

【００４３】図3のステップ３０２に示すように、パブリッシャは、まず、それが接続されているハブに、それ自体をクライアントとして登録する。次に、パブリッシャは、公表するときにそれがどの「広告」を使用するのかについて、ハブに通知する。広告は、「公表する意向」を表わし、パブリッシャがどのタイプのイベントを公表しようとしているかをハブに伝える。ステップ３０２においてハブに通知するために、パブリッシャは、既にハブにインストールしてある（図2参照）広告の名称を伝える。

【００４４】ステップ３０４において、パブリッシャはイベントを公表する時刻がくるまで待機する。上述の例におけるように、パブリッシャによっては、発生する毎に各イベントを公表するものもある。他のパブリッシャは、1日1回のように、指定した時刻まで待ち、その時刻に未公表のイベントを全て公表する。例えば、第1のパブリッシャは、人間のオペレータが売買注文をシステムに入力したときはいつでも、イベントを公表するようにしてもよい。第2のパブリッシャは時間毎に同一タイプのイベントを公表するようにしてもよい。一旦パブリッシャがイベントを作成することを決定したなら、その公表すべきイベントに相応しいフォーマットを有するＣ構造体を作成する、イベント操作ルーチンの1つをコールする（ステップ３０６において）。

【００４５】ステップ３０８において、パブリッシャは、イベント操作ルーチンの1つをコールし、未公表のイベントを公表する。これについては、以下で更に詳しく論ずることにする。ステップ３１０において、パブリッシャは、イベント操作ルーチンの1つをコールし、公表済みのイベントを廃棄する。（通常、ここに記載する実施例では、イベントの割り当てを担うあらゆるエンティティがイベントの破棄も担当する。）

【００４６】ステップ３１２において、パブリッシャは実行を中止べきか否かを判断する。中止すべきでない場合、制御はステップ３０４に戻る。中止すべき場合、ステップ３１４において、パブリッシャはハブからそれ自体を抹消し、実行を中止する。広告はハブ内に保持され、パブリッシャがそれらのイベントを実行し送出可能であることを示す。この情報は、サブスクライバへの経路を構築する際に必要となる。これについては、以下で説明する。図示していないが、同一ハブに接続されている他のパブリッシャも、図3のステップとは非同期に動作することも可能である。

【００４７】サブスクライバは、特定のイベント・タイプのイベントに予約を行う。図4は、図1のサブスクライバが、サブスクライバの処理の間に実行するステップを示すフロー・チャートである。サブスクライバは、まず最初に、それ自体をハブのクライアントとして登録す

る。次に、各予約について、「コールバック」手順を登録する。

【００４８】ステップ４０２に示すように、サブスクライバは次に、それが接続されているハブに、それ自体をクライアントとして登録する。次に、サブスクライバは、ハブを通じて受信を希望するイベント・タイプに対して「予約」を登録する。（サブスクライバは、どのタイプのイベントが広告されているかを調べるために、ハブを閲覧することができる。）予約は、イベントのタイプを指定し、更に、サブスクライバがあるタイプのイベントのみを受信することを希望していることを示す「フィルタ」も指定することができる。この場合、イベントはあるフィールドにある値を有する。

【００４９】ステップ４０４において、各予約に対して、パブリッシャは、所定の「コールバック」手順を定義する。コールバック手順は、サブスクライバがあるタイプのイベントを受信したと判断したときはいつでも、ハブによって起動される。コールバックのフォーマットは、本発明を実施する各オペレーティング・システムによって異なる場合があり、コールバック手順は、実際には、イベントを受信するときに実行されているいずれかの手順とすることができる。例えば、図４のステップ４０６において、コールバック手順は、イベントの内容を印刷することができる（そのタイプのイベントを印刷するように設計されたイベント操作手順の１つを用いて）。コールバック手順は各予約毎に定義されるので、コールバック手順は、サブスクライバが予約しようとしている各タイプのイベントに対して定義しなければならない。ここに記載する実施例では、コールバック手順に渡されたイベントは、コールバック手順が終了すると廃棄される。したがって、サブスクライバがそのイベントのいずれかの情報を保持することを望む場合、その情報をコールバック手順の中でイベント外部にコピーする必要がある。

【００５０】サブスクライバは、実行を中止する時刻がくるまで繰り返す（ステップ４０８を参照）。指定されたタイプのイベントおよび値がハブによって受信されると、このループの間、０回、１回または多数回、コールバック手順を活性化することができる。サブスクライバが予約したイベントが受信された場合、ステップ４０６においてコールバック手順を実行する。ステップ４０８において、サブスクライバが切断する時であると判断した場合、サブスクライバは任意に、その予約を抹消し、それ自体をハブから抹消し、実行を中止する。予約をハブ内に保持することによって、切断中に、以降のイベントを受信することも可能である。図示さないが、同一ハブに接続されている他のサブスクライバも、図４のステップとは非同期に、ステップを実行することも可能である。

【００５１】以下の例は、Ｃプログラミング言語で書かれており、パブリッシャおよびサブスクライバのソフトウエア例を示すものである。

【００５２】

```
/*
 *publish.c
*/
 #include <;nexus.h>;
 #include "nxs_output/order_nxs.h"
 int main()
 {
        SalesOrder *event;
        nxsClientID id;
        nxsPublicationID pub_id;
        /* Register the Client and the publication */
        nxsCreateClient ("order publisher", NULL,NULL , 0, &;id);
        nxsRegisterPublication (id, "order_advertisement", &;pub_id);
        /* Create the event */
        event  =  nxsCreate_SalesOrder();
        event->;customer_name = "Jim Nexus";
        event->;customer_address.street = "12345 Anistreet";
        event->;customer_address.city = "Mountain View";
        event- >; customer_address.state = "CA";
        event- >; customer_address.zip = "95000";
        /* Publish Event */
        nxsPublish_SalesOrder (id,pub_id,event,0,0);
        /* Clean up */
```

```
        nxsDestroy_SalesOrder(event);
        /* Unregister the publication and client */
        nxsUnregisterPublication (id,pub_id);
        nxsDestroyClient (id);
        return (0);
}
/*
   * subscribe.c
  */
#include <; nexus.h >;
#include "nxs_output/order_nxs.h"
/* Callback function to be called when each event is received */
void event_callback (
            nxsClientID id,
            nxsSubscription sub_id,
            nsxULong seq_num_high,
            nsxULong seq_num_low,
            SalesOrder  *the_event,
            void *client_data)
{
            char *st;
            printf("Event received!\n");
            st  =  nxsStringify_SalesOrder(the_event);
            printf(st);
            free(st);
}
int main(int argc, char **argv)
{
            Sales Order *event;
            nxsClient ID id;
            nxsSubscriptionID sub_id;
            /* Register Client and subscription */
            /* (Only subscribe to specific ZIP codes) */
            nxsCreateClient ("order subscriber",NULL, NULL, 0,&;id);
            nxsRegisterSubscription SalesOrder(id,
                        "customer_address.zip ==95000",
                         event_callback, NULL&;sub_id);

            /* Main loop */
            nxsMainLoop(id);
            /* Unregister subscription and client */
            nxsUnregister Subscription(id,sub_id);
            nxsDestroyClient(id);
            return(0);
 }
```

【００５３】ＩＩＩ．ハブ

図５は、図１のハブ１０６の詳細を示すブロック図である。図１のハブ１０８，１１４、そして実際ここに記載する実施例における他のハブも全て同様の構造を有する。ハブ１０６は、ネットワーク１２０を通じて（または直接）遠隔ハブ１０８，１１４に、そして０箇所以上のパブリッシャ１０２および０箇所以上のサブスクライバ１０４に接続されている。また、ハブ１０６は、ハブの管理に関する情報５３０、そのクライアント（パブリッシャおよびサブスクライバ）の管理に関する情報５３

2、およびシステム・イベント管理に関する情報５３６を受信する。

【００５４】ハブ１０６への全ての入力およびハブ１０６からの全ての出力は整列されている。例えば、遠隔ハブ１０８およびサブスクライバ１０２からハブ１０６によって受信されたイベントは、ハブ１０６がそれらを処理する時間を有するまで、各イベント・キュー５０２，５０４の中に、イベントの優先度順に格納される。更に他の例として、ハブ１０６が遠隔ハブ１１４およびサブスクライバ１０４に送るイベントは、ハブ１０６がそれらを送出する時間を有するまで、メモリ内の各イベント・キュー５０６，５０８の中にイベントの優先度順に格納される。ハブは、先入れ先出し方式を用いて、イベントの分配を行う。これが意味するのは、同一優先度レベルを有する全てのイベントは、それらがパブリッシャから受け入れられた順に、ハブ１０６によって送出されるということである。優先度レベルが高い全てのイベントは、それより優先度が低い待機中イベントよりも早く送出される。ここに記載する実施例は、イベントがシステム中を移動する際、単一のパブリッシャからの同様のイベントの順序付けを維持することを保証する。これは、例えば、更新の順序を保持しなければならないトランザクションの処理では、重要である。パブリッシャ間の順序付けは、経路や可用性によって異なるので、保証されていない。

【００５５】また、図5はハブ１０６内部の、以下にあげる機能ブロックも示す。即ち、クライアント管理ブロック５１０、イベント管理ブロック５１２、プリプロセッサ・ブロック５１４、格納ブロック５１６、マッチ・ブロック５１８、および遠隔管理ブロック５２０である。クライアント管理ブロック５１０は、クライアントの登録および抹消を扱う。イベント管理ブロック５１２は、新たな接続、タイプ、経路、広告、および予約のようなシステム・イベントの管理を扱う。プリプロセス・ブロック５１４は、エンベロープ情報のイベントへの追加のようなタスクを実行する。格納ブロック５１６は、ハブのメモリである。マッチ・ブロック５１８は、経路情報をイベントに追加し、イベントのルーティング(routing)に関して以下に述べるような方法でイベント・フィルタ処理を行い、他に接続されている適切なハブにイベントを送出する。遠隔管理ブロック５２０は、システム管理タスクを実行する。

【００５６】図8は、図1のハブ１０６のメモリ６９０に格納されるデータ構造の図であり、図5の機能ブロック全てによってアクセス可能である。データ構造は、インストールされた広告のテーブル７００、登録済みクライアントのテーブル７３０、登録済み公表のテーブル７６０、および登録済み予約のテーブル７７０を含む。データは、パブリッシャおよびサブスクライバが図２、図３、および図４の種々のステップを実行する際に、以下のように図８のデータ構造の中に配置される。

【００５７】図2のステップ２０２は、インストール済み広告テーブル７００のエントリを満たす。

【００５８】図3および図4のステップ３０２，４０２は、登録済みクライアント・テーブル７３０のエントリを満たす。ユーザ名フィールド７３４およびユーザ・パスワード・フィールド７３６はＮＵＬＬ値を有すると、現ユーザＩＤを使用すべきことを指示することができる。ハブ名７３７およびテリトリ名７４０は、ＮＵＬＬにセットすると、デフォルトのハブおよびテリトリの使用を指示することができる。フラグ７４２は、例えば、"call-back-on-any-thread"（いずれかのスレッドにコール・バックする）を含み、イベント・コール・バックが新たなスレッドに到来することを示す。

【００５９】図3のステップ３０２も、登録済み公表テーブル７６０のエントリを満たす。

【００６０】広告名７６４は、ハブにインストールされた広告の名称である。図4のステップ４０４も、登録済み予約テーブル７７０のエントリを満たす。内容フィルタ７７４については以下で論じる。コールバック手順の名称は、サブスクライバ内に（Ｃ　ＡＰＩによって）局所的に保持される。ＡＰＩは、予約ＩＤからのコールバックを発見する。クライアント・データ・フィールドは、これもＡＰＩによって保持され、予約に関連する情報を、各イベントのコールバックによって返すことができるようにする。戻される情報は、予約アプリケーション(subscribing application)にとって意味がある。

【００６１】以下の節では、サブスクライバが予約を登録するときに指定可能な様々なタイプの内容フィルタについて説明する。上述の例に示したように、サブスクライバは、ハブが入来するイベントの流れにフィルタをかけ、ある基準と一致したイベントのみをサブスクライバに通すことを要求できる。フィルタは、好ましくは、予約を登録するためのルーチンにパラメータとしてサブスクライバが送った式の文字列(expression string)として指定される。ここに記載する実施例における内容フィルタに対する式の文字列のシンタックスは、以下の通りである。（勿論、いずれの適切なシンタックスも使用可能である）

【００６２】

記号	意味	許されるタイプ
＝	等しい	全ての基本タイプ
！＝	等しくない	全ての基本タイプ
＞	より大きい	数値および文字列タイプ
＞＝、＜＝	以上	数値および文字列タイプ

and	論理積	式またはブーリアン
or	論理和	式またはブーリアン
not	論理否定	式またはブーリアン

【００６３】イベント属性名は、イベント内のどのフィールドを比較すべきかを指定するために用いられる。サブ・フィールド（構造内部のサブフィールド）は、ドット表記(dot notation)を用いて指定する。名称は、Ｃプログラミング言語に精通している人には既知であるが、列挙タイプ(enumerated type)とすることも可能である。一旦内容フィルタを指定したなら、イベント・ルーティング(event routing)の間使用される。これについては、以下で図９ないし図１３に関連付けて説明する。予約に対して各内容フィルタを記述する情報は、図８のフィールド７７４に格納される。

【００６４】図６は、図５のハブ１０６のメモリに格納されるデータ構造を示す図である。全てのハブは同様のデータ構造を収容する。ここに記載するインプリメンテーションは、分散型オブジェクト・モデルを使用するが、あらゆる適切な組織的構造が使用可能である。図６のデータ構造の詳細を図７に纏めておく。理想的なのは、図６のデータ構造がシステム内の全広告を記述し、近隣のハブに、クライアント・ハブがあるタイプのイベントをどこに送出すべきかを示すことである。図１１および図１３は、それぞれ、図６のデータ構造を移植するステップ、および図７のデータ構造を用いてハブ間でイベントをやりとりするステップを記述するものである。図１２は、データを移植されたデータ構造の一例を示す。

【００６５】ハブ１０６（「クライアント・ハブ」）は、サブスクライバである現ハブの各クライアント（「クライアント」６８４および「S」６９５によって示す）と、現ハブに接続されている各近隣のハブ（「近隣ハブＡ」６９６および「近隣ハブＢ」６９７）を表わす、データ・オブジェクトを含む。現ハブは、その近隣ハブの予約オブジェクトを追跡し続ける。予約オブジェクト「Ｓ」６５０は、各近隣ハブを通じて到達可能な全てのサブスクライバを表わす。各近隣オブジェクトは、"mycost"および"theircost"という２つの関連するコスト値を有する。"mycost"は現ハブから近隣ハブに情報を送るコストを表わす。"theircost"は、近隣ハブから現ハブに情報を送るコストを表わす。"their cost"は、以下で説明するように、経路を定義するために用いられる。予約側ハブは、「公表側」ハブに、最も低い"theircost"でイベントの転送(deliver)を依頼する。"mycost"は、当該リンクを通じて情報を送るためには、現ハブにどの位のコストが必要かを、近隣ハブに通知するためのみに使用することが好ましい。例えば、コストは、財政的コスト、単位時間のコスト、便利さの尺度としてのコスト、またはその他のあらゆる適切なコストを表わすことができる。

【００６６】ハブ１０６は、システムに登録されている各広告について（"RemoteAd"）オブジェクトを含む（即ち、公表する意向毎。図２のステップ２０２参照）。各RemoteAdオブジェクトは、１つ以上の"AdSource"オブジェクトを指し示す。各AdSourceオブジェクトは、現ハブと、広告のパブリッシャが常駐するハブとの間のパス(path)を表わす。各Adsourceオブジェクトは、その広告の最初のパブリッシャへのパスに伴うコストを格納する。したがって、各AdSourceに対する「コスト」値は、イベントをそのソースから現ハブの近隣ハブまで、またはその反対に現ハブまで送るための全コストを含む。

【００６７】各AdSourceオブジェクトは、関連する「シンク・オブジェクト」(sink object)のリストを有する。（SinkCaおよびSinkNbは、単一のリストに位置付けることが好ましいが、図ではそのように示されていない。）各シンク・オブジェクトは、関連する予約のリストを有する。例えば、SinkCaは、RemoteAd５１０の広告に一致する、クライアント６９４の全予約のリスト５０５を有する。同様に、SinkNbは、RemoteAd５１０の広告と一致した、近隣ハブＢの全予約（および近隣ハブＢを通じて到達可能な全予約）のリスト５１５を有する。

【００６８】各近隣オブジェクトは、パブリッシャと当該近隣オブジェクトとの間の全パスについてAdSourcesを指し示すAdSourceRefリストを有する。これらの経路は、以下で説明するように、ハブ間でシステム・イベントの経路を決定する際に用いる。

【００６９】ＩＶ．ハブ間におけるイベントのルーティング

図１１は、図１のハブ１０６，１０８，１１４が図６のデータ構造に移植する際に実行するステップの詳細を示すフロー・チャートである。これらのステップは、ネットワークを最初に初期化するときに実行する。続いて、種々のハブが「システム・イベント」を発行し、ハブ接続、イベント・タイプ、広告、ルーティング、予約等において変更があったことを、他のハブに伝える。追加資料Ａは、本明細書の一部であり、この言及により本願にも含まれるものとするが、ここに記載する実施例においてハブ間で渡すことができるシステム・イベント例の概要を含む。

【００７０】図１１において、フロー・チャート内のステップに影響を与える、追加資料Ａのシステム・イベントの名称を、ステップの次に示す。ステップ１００２では、ハブが互いにシステム・イベントを送出し合い、ネットワーク内のハブ間における物理的接続を定義する。各ハブは、その近隣ハブへの物理的接続を表わす近隣オブジェクトを作成する（図６参照）。ステップ１００４において、ハブは互いにシステム・イベントを送出し、

広告を公表可能なイベントのタイプを定義する。各ハブは、全てのイベントを知っていることを確認する。このステップは、ハブをテリトリに追加するときに実行される。ステップ１００６において、パブリッシャに接続されているハブが互いにシステム・イベントを送出する。これらはハブ間で転送され、当該ハブ間で広告を分散する。例えば、追加資料Aのシステム・イベント"NSX_SYS_EVENT_CREATE_NEW_ADVERTISEMENTは、ハブにおいて、RemoteAdオブジェクトを作成させる（図6参照）。

【００７１】ステップ１００８において、パブリッシャに接続されているハブは、他のハブにシステム・イベントを送出する。このシステム・イベントはハブ間で転送され、パブリッシャのハブから他のハブへの経路を決定する。（システム・イベント"NXS_SYS_EVENT_NEW_ROUTES"により、ハブ内にAdSourceオブジェクトを作成する（図6参照））。経路の存在を登録するために、最初のハブが、広告名／ＩＤ、その（パブリッシャハブの）名称およびテリトリ、ならびにイベントをパブリッシャハブから現ハブまで送出する際にかかる費用を含む、システム・イベントをその近隣ハブに送出する。ここに記載するインプリメンテーションでは、コストは初期状態では0に設定される。各近隣ハブはこのプロセスを繰り返し、当該ハブがイベントを受信した際の接続のコストを加算する。各ハブでは、これがそのハブへの最初の経路である場合、ローカル・クライアントをチェックし、予約との一致を調べる（この動作の結果、"NEW_SUBSCRIPTION"システム・イベントが、"NEW_ROUTES"の送出元に得られる。転送のために、各経路を別個に検討する。経路は、それが受信されたハブ、発信ハブ(oroginating hub)、またはイベントを既に知っているハブには転送されない（以下で説明するように、ルーティング・リストから判断する）。経路は、RemoteAdに対する最初の経路である場合、またはそれが２番目の経路であり、目的地のハブが現行の経路提供元である場合にのみ、他のハブに転送される。

【００７２】ステップ１０１０において、あるサブスクライバに接続されているハブが他のハブにシステム・イベントを送出し、これら他のハブが知っている広告に対する予約を登録する。（システム・イベント"NXS_SYS_EVENT_NEW_SUBSCRIPTIONS"は、他のハブに予約を送出する。図6参照）。このシステム・イベントを受信したハブは、予約オブジェクト（図6参照）を作成し、それについてのRemoteAdを伝える。最少コストのパスを有するAdSourceのことを、現AdSourceと呼ぶ。RemoteAdは、現AdSourceに伝え、Neighborオブジェクトのシンク・オブジェクトおよび予約オブジェクトを作成する（シンク・オブジェクトおよび予約オブジェクトが存在しない場合）。次に、シンクが、予約オブジェクトに対するSubscriptionRef エントリ（例えば、５０５，５１２）を追加する。他のハブから広告が受信された場合、現ハブは、新たな予約に対するシステム・イベントを、現最少コスト経路の一部である近隣ハブに転送する。

【００７３】この予約の他のハブへの転送は、本発明の重要な態様である。これは、本質的に、サブスクライバのハブからパブリッシャのハブに戻る予約の連鎖を作成する。この連鎖は、パブリッシャとサブスクライバとの間の最少コスト経路に従う。

【００７４】図１２は、サブスクライバのハブからパブリッシャのハブへの予約連鎖の作成例を示す。図１２は５つのハブＡ，Ｂ，Ｃ，Ｄ，Ｅを示す。ハブＡにはパブリッシャＰが接続されている。ハブＣにはサブスクライバＳが接続されている。この例では、各接続は、「１」のコストを有する。したがって、例えば、ハブＡおよびＣ間には、全コスト＝２の経路があり、ハブＡおよびＣ間には全コスト＝３の経路がある。

【００７５】図１２は、図１１のステップ１０１０の後のハブのステータスを示す。接続、イベント・タイプ、および経路がハブ間に分散され、RemoteAd，AdSource，Neighbor，およびAdsourceRefオブジェクトが各ハブ内で作成されている。

【００７６】この例では、サブスクライバＳがハブＣ上で予約を登録する（Ｓがそれ自体をクライアントとして登録したときに、ハブＣは既に予約オブジェクト１１０２を作成しており、サブスクライバＳがハブＣのクライアントであることを示している）。次に、ハブＣは、「新たな予約システム・イベント」（「予約」）を、当該イベントに対する広告のパブリッシャへの最少コスト上にある、近隣ハブに送信する。この例では、ハブＢからパブリッシャ・ハブＡまでのコストが「１」なので、ハブＣは予約をハブＢに送信する。ハブＢが予約を受け取ると、ハブＣに対する近隣オブジェクトに接続されている予約オブジェクト１１０４を追加し、サブスクライバがハブＣまで到達可能であることを示す。

【００７７】次に、ハブＢは、イベントに対する広告のパブリッシャへの最少コスト経路上にある、近隣ハブに予約を送信する。本例では、ハブＡからハブＢへのコストが「０」であるので、ハブＢは予約をハブＡに送信する。ハブＡが予約を受信すると、ハブＢに対する隣接オブジェクトに接続された予約オブジェクト１１０６を作成し、予約がハブＢまで到達可能であることを示す。ハブＡは、それ自体が予約されつつある広告の作成元ハブであるので、これ以上予約を送信しない。

【００７８】一旦図１２のデータ構造が確立されると、パブリッシャＰが公表したイベントは、予約オブジェクト１１０６，１１０４，１１０２によって定義されたパスに沿って、サブスクライバＳに到達するまで、ハブ間で送出される。

【００７９】図１３は、図１のハブが、公表したイベントをシステムのハブを通じて送信することによって、サブスクライバにそのイベントを送出する際に実行するス

テップの詳細を示すフロー・チャートである。図13のステップは、1）それ自体に接続されているパブリッシャからのイベントを受信したハブ（ステップ1202）、または2）近隣のハブからのイベントを受信したハブ（ステップ1203）によって実行される。図13のステップは、各イベントがシステム全体にわたって送信される際に、種々のハブによって実行される。以下の節では、図13のステップを実行するハブのことを、「現ハブ」と命名する。

【0080】現ハブがそれに接続されているパブリッシャの1つからのイベントを受信すると（ステップ1202）、現ハブのプリプロセッサ514は、最初に「エンベロープ」をそのイベントに加える（ステップ1204）。イベント・エンベロープのフォーマットについては、図9に関連付けて以下で説明する。プリプロセッサ514は、次にステップ1204において、このイベントのイベント・タイプがハブに登録されているか否か判断する。登録されていなければ、パブリッシャはこのイベントを公表することができない。登録されていれば、ステップ1206において、ハブはそのイベントを、ハブの入来イベント・キューに入れて、更に処理を行う。ここに記載する実施例では、イベント・キューは重複するイベントを破棄する（イベント・エンベロープの連番またはタイム・スタンプによって判断する）。

【0081】現ハブが近隣ハブからイベントを受信した場合（ステップ1203）、近隣ハブはそのイベントを、現ハブの入来イベント・キューに入れる。

【0082】ステップ1208において、現ハブのフィルタ518が、現ハブの入来イベント・キューからそのイベントを取り出す。次に、フィルタ518は、キュー内で費やした時間量を示す、エンベロープの経過時間フィールドを増分する。ステップ1210において、フィルタ518は、ルーティング・ブロックをイベントに加える。イベントを送信する各ハブは、他のルーチング・ブロックを当該ブロックに加える。2つのルーティング・ブロックのフォーマットを図10に示す。

【0083】ステップ1212において、現ハブは、特定のAdSourceから来るイベントに関心がある全てのサブスクライバを表わす、「現」AdSourceオブジェクトの位置を突き止める。これは、通常、現ハブとサブスクライバ間の最少コスト・パスを表わす（ここに記載する実施例は、一旦AdSourceデータ構造が確立されたなら、再送信は行わないことに注意されたい）。AdSourceオブジェクトの位置を突き止めるには、現ハブに対する全RemoteAdを探索するか（イベントが、現ハブに関係するパブリッシャから来た場合）、あるいは近隣ハブに対するAdSourceRefリストを探索する（イベントが近隣ハブから来た場合）。ステップ1216～1222は、1つ以上のサブスクライバにイベントを送出する。ステップ1216～1222は、AdSourceに関係する各シンク・オブジェクト毎に実行される。したがって、ステップ1216～1222は、送信されたタイプのイベントを受信することを要求した各サブスクライバのために実行される。

【0084】ステップ1216において、イベントの存続時間がエンベロープの経過時間よりも小さい場合、このイベントは消滅し、これ以上現ハブによって送信されない（ステップ1218）。その他の場合、ステップ1220において、フィルタ518は、イベントが、サブスクライバが指定したパラメータに該当する内容を有するか否かを判断する。サブスクライバは、イベント・タイプに予約した時点で、内容フィルタを指定している（図8のフィールド774を参照）。例えば、あるサブスクライバは、ロサンゼルスに住んでいるカストマのためのSalesEventのみを受信することを要求した可能性がある。サブスクライバが指定した範囲の値をイベントが有する場合、ステップ1222においてイベントはサブスクライバに送出される（以下で説明するように、直接またはそのハブを通して）。各サブスクライバは異なる内容フィルタを指定した可能性がある（あるいは、内容フィルタを指定しない可能性もある）。例えば、第1のサブスクライバが、カストマが住んでいるロサンゼルスのSalesEventの受信希望を指定し、第2のサブスクライバが、カストマが住んでいるサンフランシスコのSalesEventの受信希望を指定した可能性がある。この例では、イベントは第1のサブスクライバとは一致するが、第2のサブスクライバとは一致しない。

【0085】ステップ1222において、フィルタ518は、一致したサブスクライバにイベントを送出する。サブスクライバは、現ハブのクライアントであることもあり、その場合イベントは直接サブスクライバに送出される。あるいは、サブスクライバは近隣のハブに、直接的にまたは間接的に接続されている場合もある。一致したサブスクライバが近隣のハブに接続されている場合、現ハブはイベントをその近隣ハブに送信する（エンベロープのフィールド802から判断して、その近隣ハブがイベントを発生したハブでない場合。図9参照）。また、現ハブは、そのイベントを既に見たハブにも、当該イベントを送信しない（イベントに添付されているルーティング・ブロックから判断する。図10参照）。

【0086】ここに記載する実施例では、現ハブは、近隣ハブが1つ以上の一致する予約を有していても、この近隣ハブにイベントを送信するのは1回のみである。したがって、図6の予約オブジェクト651がイベントと一致した場合、そのイベントを近隣ハブBに転送する。予約オブジェクト652もそのイベントと一致する場合、近隣ハブBに2回目のイベント送出を行う必要はない。近隣ハブBがこのイベントを受信すると、これも図13のステップを実行し、そのクライアントのサブスクライバのいずれかがイベントと一致するか否かを判断する。

【００８７】図９は、イベントのエンベロープ・データ構造のフォーマットを示す。図５のプリプロセッサは、イベントが公表される前に、そのサブスクライバから受信したイベントにエンベロープを加える。エンベロープは、各イベントに付随し、起点ハブ８０２、広告名８０４、パブリッシャ名８０６、テリトリ名８０８、初期タイム・スタンプ８１０、存続時間８１２、優先度８１４、フラグ８１６、および経過時間フィールド８２２を含む。フラグは、ここに記載する実施例では、使用しない。別の実施例では、フラグは、例えば、イベントが「永続的」か、あるいは「一時的」かを示す場合もある。

【００８８】図１０は、イベントの２つのルーティング・ブロックのフォーマット例を示す。各ハブは、それがそれ自体を通じて送信するイベントに、ルーティング・ブロックを追加する。ＡＰＩは、イベントがあるサブスクライバに到達するために取った実際の経路についての情報を返す、get_current_route_infoルーチンも含む。ルーティング情報は、各訪問ハブ毎に１つのエントリで、リンクされたリストとして、システムによって格納される。各エントリは、現ハブに対する出立タイム・スタンプ、現ハブのハブ名、および現ハブのテリトリ名を含む。また、任意にルーティング情報は、パブリッシャによってイベントに最初に割り当てられる、６４ビットの連番を含む。

【００８９】Ｖ．テリトリ

図１の各テリトリ１３０，１４０は、共通のイベント辞書を有するハブの集合である。テリトリは組織を反映し、地理や直接物理的接続性(direct physical connectivity)に結びつける必要はない。ここに記載する実施例では、１テリトリ内のハブは全て完全に接続されていなければならない。なぜなら、広告情報はテリトリ間を跨いで転送されないからである。したがって、あるテリトリ内の２つのハブ間には、このテリトリを離れる必要なく、間接的な場合もあるが、常にいくつかの経路がある。好適実施例では、テリトリの階層が存在し、レベルが低い方のテリトリはそれらの親テリトリのイベント辞書を継承する。例えば、ある会社が、各支店に１つのテリトリを有するように選択を行うことがあり、これら全てがある「社内イベント」を理解している。

【００９０】１つのハブは、１つのテリトリのみに属することが望ましい。クライアント（パブリッシャまたはサブスクライバ）がハブに接続するとき、どのテリトリまたはテリトリにそれが属するかを指定する。異なるテリトリでは、イベント、例えば、SalseOrderの意味が異なる場合があるので、クライアントは１つのテリトリに属さなければならない。ここに記載する実施例では、各ハブは所定のデフォルトテリトリを有し、アプリケーション開発者がテリトリを指定しない場合、これが用いられる。

【００９１】別の実施例では、多重テリトリハブが会社間通信には特に有用である。２つの会社が、ここに記載する本発明の実施例を使用して通信を行うことを決定した場合、共有すべきいくつかのイベント・タイプおよびフォーマットについて合意し、共同テリトリ(joint territory)を設定することができる。テリトリ情報は広告の一部として保持されるので、２つの会社を接続するハブは、この共同テリトリのイベントのみを転送する。これについては、イベントのルーティングに関連付けて以下で説明する。

【００９２】ＶＩ．データベース接続性 (database connectivity)

図１４は、図１のネットワーク１２０に組み込まれたデータ・ベース・アプリケーション１３０２のブロック図である。データベース・アプリケーション１３０２は、インターフェース・ルーチン１３０４を有するデータベース・ライブラリを含み、ユーザがデータ・ベース１３０８内の情報にアクセスできるようにする。データ・ベース１３０８は、例えば、オラクル(Oracle)、サイベース(Sybase)、またはインフォミックス(Informix)で作成されたデータ・ベースとすることができる。本発明は、トランザクション・モニタと共に動作可能な適切なデータ・ベースであれば、どれを用いても実施可能である。

【００９３】図１４は、トランザクション・モニタ１３０６も含む。トランザクション・モニタ１３０６は、トランザーク・コーポレーション(Transarc Corporation)が製造するエンシーナ・トランザクション・モニタ(Encina transaction monitor)であることが好ましい。また、例えば、ノベル(Novell)が製造するツクセド・トランザクション・モニタ(Tuxedo transaction monitor)とすることもできる。あるいは、適切なトランザクション・モニタのいずれかを用いて、本発明を実施することができる。データ・ベース・ライブラリ１３０４を有するデータ・ベース接続部１３１０は、ネットワーク１２０に接続し、ネットワーク１２０とのデータ送受信を行う。データ・ベース接続部１３１０は、ハブ機能を実行するソフトウエアを含んでもよく、あるいは別個のハブに接続してもよい。図１４において、データ・ベース・アプリケーション１３０２はデータ・ベース１３０８に情報を送出する。データ・ベース・アプリケーション１３０２も、データ・ベース１３０８からの情報を要求し、その情報を人が読むことのできる形態で表示する。

【００９４】データ・ベース接続部１３１０は、パブリッシャ、サブスクライバ、または双方とすることができる。データ・ベース接続部１３１０がパブリッシャである場合、例えば、データ・ベース１３０２において値の変更がある毎に、イベントを公表する。また、例えば、規則的な時間間隔でイベントを公表し、最後の公表以降データ・ベースにおいて更新されたデータを反映することも可能である。データ・ベース１３０２がサブスクラ

イバである場合、受信したイベントの情報をデータ・ベース１３０２内に格納する。ここに記載する実施例は、データ・ベース接続部１３１０用のコンフィギュレーション・ファイル（図示せず）を含む。コンフィギュレーション・ファイルは、例えば、データベース接続がパブリッシャか、サブスクライバか、あるいは双方かを指定する。また、コンフィギュレーション・ファイルは、例えば、公表すべきおよび／または予約すべきイベントの記述も含む。また、コンフィギュレーション・ファイルは、データ・ベース１３０８のレイアウトに関する情報も指定する。

【００９５】図１５は、データ・ベースがサブスクライバであるときに実行されるステップを示すフロー・チャート１４００である。例えば、ネットワーク１２０からイベントを受信すると、これらイベントからのデータがデータ・ベース１３０２に追加される。フロー・チャート１４００は、データ・ベース接続部１３１０が既にそれ自体およびその予約（群）を、上述のように、登録済みであると仮定する。ステップ１４０２においてデータ・ベース接続部１３１０がイベントを受信すると、データ・ベース接続部は、そのイベント内のデータを、データ・ベース１３０６内にマップし格納する。データがデータ・ベース内に完全に格納されるまでは、このデータはデータ・ベースには「受け入れられて」いない。「受け入れ」(commitment)とは、データが実際にデータ・ベースに入力されることを意味する。データへの変更が受け入れられるまで、これらは一時的格納場所に保持される。したがって、「受け入れ」処理が実行されるまでは、受け入れられていないデータは、いつでも「ロールバック」(rolled back)することができ、データ・トランザクションを取り消すことができる。「ロールバック」の間、一時的格納部におけるデータの変更は廃棄され、データ・ベースは、変更が行われた前のその状態に保持される。状況によっては、データ・ベース接続部１３１０は、あるイベントを受信するまでは、「受け入れ」コマンドを発行しない。

【００９６】図１６は、データ・ベースがパブリッシャである場合に実行されるステップを示すフロー・チャート１５００である。例えば、人間のユーザがデータの追加、削除、および修正をデータ・ベース１３０２において行うと、データ接続部１３１０はネットワーク１２０に周期的なイベントを送出する。ステップ１５０２において、トランザクション・モニタ１３０６は、データ・ベース接続部１３１０が「ノード」であることの通知を受ける。このステップを実行する正確な方法は、トランザクション・モニタ１３０６の製造者によって異なるが、一般的にこのステップは当業者にはよく知られている。ステップ１５０４において、前述のように、データ・ベース接続部１３１０は、それ自体およびその広告（群）を登録する。

【００９７】ステップ１５０６～１５１２は、例えば、ユーザがデータ・ベース１３０８に変更を加える場合に実行されるステップを示す。ステップ１５０６においてデータ・ベース・トランザクションが開始すると、データ・ベース１３０８は、トランザクション・モニタに、例えば、Ｘ／ＯｐｅｎのＸＡプロトコルを用いて、トランザクションが開始したことを通知する。トランザクション・モニタ１３０６は、データ・ベース接続部１３１０に、トランザクションが発生していることを通知し、データ・ベース接続部１３１０はトランザクションに注目する。ステップ１５０８において、ユーザがデータ・ベースに変更を加えるが、ユーザは変更を受け入れていない）。

【００９８】ステップ１５０７において、ユーザ（または、変更を加えるソフトウエア）が、トランザクションを受け入れるべきことを指示すると、ステップ１５１０において、この事実はトランザクション・モニタ１３０６に伝達される。トランザクション・モニタ１３０６はデータ・ベース接続部１３１０に通知し、受け入れられたデータをデータ・ベース１３０８から引き出し、この受け入れられたデータを含むイベントを公表する。イベントは図１３に示すように公表される。ステップ１５１２においてユーザ（または、変更を加えるソフトウエア）が、トランザクションをロールバックすることを指示すると、この事実はトランザクション・モニタ１３０６に伝達される。トランザクション・モニタ１３０６がデータ・ベース接続部１３１０に報告すると、データ・ベース接続部１３１０はステップ１５０６の変更について「忘れ」、イベントを公表しない。このように、ここに記載する実施例は、商用データ・ベース処理には一般的な受け入れおよびロールバック処理を考慮に入れている。

【００９９】別の実施例において、使用するデータ・ベース生産物１３０２がトランザクション・モニタを含まない場合、データ・ベース１３０８と接続するときは、データ・ベース接続部１３１０がトランザクション・モニタとして機能する。更に、データ・ベース接続部１３１０は、このコンフィギュレーションでは、パブリッシャおよび／またはサブスクライバとしても機能する。データ・ベース１３０８は、それが実際にはデータ・ベース接続部１３１０に接続されていても、トランザクション・モニタに接続されていると考える。

【０１００】ＶＩＩ．要約

要約すると、本発明の好適実施例は、情報を公表する複数の「パブリッシャ」エンティティと、情報を要求し使用する複数の「サブスクライバ」エンティティとを含む。パブリッシャおよびサブスクライバは、ネットワークを通じて互いに接続されている。ネットワークは、そのルーティングが「内容を基本とした」、「格納および転送」ネットワークである。

【0101】本発明の上述の実施例では、情報の基本単位を「イベント」と呼ぶ。パブリッシャはイベントを公表し、サブスクライバは、当該サブスクライバが定義した基準と一致するイベントに予約する。公表および予約は非同期に行われる。パブリッシャおよびサブスクライバは、直接互いについての知識を有していない。システムは、パブリッシャから公表されたイベントを受信し、そのイベントを全ての適切なサブスクライバに送信する。各サブスクライバは、当該サブスクライバが指定した予約基準と一致する場合のみ、システム上で公表された全てのイベントを受信することが保証されている。

【0102】上述の本発明の実施例は、レガシ・システム、レガシ・アプリケーション、およびレガシ・ハードウエアをシステムに容易に統合し、ユーザがそのシステムを使用するために修得しなければならない情報量を最少に抑えようとするものである。レガシ・データ・ベースは、データ・ベース接続部によって、ネットワークに予約することができる。この場合、データ・ベース接続部は、パブリッシャ、サブスクライバ、または双方とすることができる。

【0103】ここに開示した本発明の明細書および実施を検討することから、他の実施例も当業者には明白であろう。明細書および例は例示としてのみ解釈し、本発明の真の範囲は特許請求の範囲によって示されるものであることを意図するものである。

【0104】追加資料A

（44頁）

6．0　システム・イベント

ハブ間で通信を行うために23個のシステム・イベントが用いられる。これらは、通常のイベントのヘッダ構造のいくつかを共有するが、その他の場合、通常のイベント・コード・パスとは完全に異なり、その外側で処理される。システム・イベント構造は、include/nxs_syspackP.hhの中で定義される。

```
typedef struct nxsSystemEventHeader{
                    nxsULong length;
Byte length of the event
                    nxsLong magic_number;          Used to det
ect bogus evnets
                    nxsUlong nexus_version;
 Nexus version number
                    nxsULong routing_offset;
 Nexus version number
                    nxsULong routing_offset;
 nxsEventRoutingEntry
                    nxsULong sys_event_type;
    NXS_SYS_EVENT_*
                    nxsULong data_offset:
                      String data
    }  nxsSystemEventHeader;
```

sys_event_typeの値は、transport/hub/sysevent.hhの中で見つけることができる。全てのシステム・イベント処理コードは、nexushub.hhにおいて宣言され（NexusHubの方法として）、sysevent.ccにおいて定義される。システム・イベント・データは、離間された別個のトークンで構成されたストリングである。可能なトークンは以下の通りである。string　<;space>;および<;nul>;以外のあらゆる文字のストリング

/string/　^_(／037)によって囲まれた^_(／037)を除くあらゆる文字のストリング

long　符号付き32ビット値

ulong　符号なし32ビット値

（45頁）オブジェクト参照はストリング形式で転送する。システム・イベントは、いくつかの通信スタイルに用いられる。接続の確立の間、システム・イベントは、ピア間メッセージ(peer-to-peer message)として用いられる。これらのイベントは、他のハブには見えない。他のイベントの殆どは、テリトリの状態に影響を与え、テリトリグラフ(territory graph)全体に転送される。1つのハブからそれ自体に数個のイベントを送ることができる。

```
NXS_SYS_EVENT_CONNECT_HUB
        /string/        Sending hub name
        /string/        Sending hub territory
        string            Sending hub imcoming EventQueue:: PushConsu
mer
         ulong            Receiving hub's cost
```

ulong　　Sending hub's cost

このイベントは、NexusAdmin::HubAdmin::connectHubへのコールの結果として送出される。送出側のハブは既に接続のために、NeighborおよびEventQueueを作成してある。受信側のハブもシステム・イベントを受信したときに、同様のことを行う。尚、送出側ハブは既に、NexusAdmin::HubAdmin::connectHubへのアーギュメントを通じて、受信側の入来EventQueue::PushConsumerを有することに注意されたい。これが受信側ハブにとって最初の接続である場合（テリトリを連結している）、テリトリの共有データ（イベント・タイプ等）を獲得する必要がある。

NXS_SYS_EVENT_CONNECT_FAILED

/string/　　Sending hub name
/string/　　Sending hub territory
/string/　　Reason for connection failure

接続の確立ができなかったときに送る。データ構造を消去する。

NXS_SYS_EVENT_DISCONNECT_HUB

/string/　　Sending hub name (or neighbor name)
/string/　　Sending hub territory

このシステム・イベントは、それ自体または他のハブに送ることができる。NexusAdmin::HubAdmin::disconnectHubへのコールの結果として、それ自体に送る。この場合、最初のアーギュメントは近隣ハブの名称である。ハブは同様のイベントを近隣（送出側ハブの名称を含む）に送出し、接続遮断を行う。これによって、次のシステム・イベントが送出される。NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS, NXS_SYS_EVENT_DELETE_ROUTES, および NXS_SYS_EVENT_DELETE_ADVERTISEMENTS。一旦接続消去イベントが送出されると、NXS_SYS_EVENT_END_OF_STREAMが伝えられる。

（46頁）イベント・シーケンスは、NXS_SYS_EVENT_DISCONNECT_HUBを受信した近隣ハブ上で同一である。

NXS_SYS_EVENT_NEW_EVENTTYPE_FILES

long　　Number of files F
long　　Number of types T
　　Repeated F times:
/string/　　.nexus file contents
/string/　　.ifr file contents
　　Repeated T times:
string　　Event type name

IFRおよびNexusをIFRにロードし、与えられたイベント・タイプを作成する。イベントは、送信元および既にそのイベントを見た近隣(neighbors)を除いて、全ての接続されている近隣に送る。

NXS_SYS_EVENT_FORWARD_EVENTTYPE_FILE

ulong　　Address of EventTypeFile state
long　　Number of event types T
　　Repeated T times:
string　　Event type name

イベント・タイプ・ファイルおよびイベント・タイプを全ての接続されている近隣に送出する。イベント・タイプが作成されたときまたは更新されたときは、それ自体にのみ送出する。

NXS_SYS_EVENT_END_OF_STREAM

/string/　　Sending hub name
/string/　　Sending hub territory

接続上で見られる最後のイベントを示す。データ構造を消去する。

NXS_SYS_EVENT_REQUEST_EVENTTYPES

/string/　　Sending hub name
/string/　　Sending hub territory

NXS_SYS_EVENT_NEW_EVENTTYPE_FILESを用いて、送出側にイベント・タイプを送出する。

NXS_SYS_EVENT_REQUEST_ADVERTISEMENTS

/string/　　Sending hub name
/string/　　Sending hub territory

このイベントは、NexusAdmin::HubAdmin::connectHubへのコールの結果として送出される。送出側のハブは既に接続のために、NeighborおよびEventQueueを作成してある。受信側のハブもシステム・イベントを受信したときに、同様のことを行う。尚、送出側ハブは既に、NexusAdmin::HubAdmin::connectHubへのアーギュメントを通じて、受信側の入来EventQueue::PushConsumerを有することに注意されたい。これが受信側ハブにとって最初の接続である場合（テリトリを連結している）、テリトリの共有データ（イベント・タイプ等）を獲得する必要がある。

NXS_SYS_EVENT_CONNECT_FAILED

/string/　　Sending hub name
/string/　　Sending hub territory
/string/　　Reason for connection failure

接続の確立ができなかったときに送る。データ構造を消去する。

NXS_SYS_EVENT_DISCONNECT_HUB

/string/　　Sending hub name (or neighbor name)
/string/　　Sending hub territory

このシステム・イベントは、それ自体または他のハブに送ることができる。NexusAdmin::HubAdmin::disconnectHubへのコールの結果として、それ自体に送る。この場合、最初のアーギュメントは近隣ハブの名称である。ハブは同様のイベントを近隣（送出側ハブの名称を含む）に送出し、接続遮断を行う。これによって、次のシステム・イベントが送出される。NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS, NXS_SYS_EVENT_DELETE_ROUTES, および NXS_SYS_EVENT_DELETE_ADVERTISEMENTS。一旦接続消去イベントが送出されると、NXS_SYS_EVENT_END_OF_STREAMが伝えられる。

（46頁）イベント・シーケンスは、NXS_SYS_EVENT_DISCONNECT_HUBを受信した近隣ハブ上で同一である。

NXS_SYS_EVENT_NEW_EVENTTYPE_FILES

long　　Number of files F
long　　Number of types T
　　Repeated F times:
/string/　　.nexus file contents
/string/　　.ifr file contents
　　Repeated T times:
string　　Event type name

IFRおよびNexusをIFRにロードし、与えられたイベント・タイプを作成する。イベントは、送信元および既にそのイベントを見た近隣(neighbors)を除いて、全ての接続されている近隣に送る。

NXS_SYS_EVENT_FORWARD_EVENTTYPE_FILE

ulong　　Address of EventTypeFile state
long　　Number of event types T
　　Repeated T times:
string　　Event type name

イベント・タイプ・ファイルおよびイベント・タイプを全ての接続されている近隣に送出する。イベント・タイプが作成されたときまたは更新されたときは、それ自体にのみ送出する。

NXS_SYS_EVENT_END_OF_STREAM

/string/　　Sending hub name
/string/　　Sending hub territory

接続上で見られる最後のイベントを示す。データ構造を消去する。

NXS_SYS_EVENT_REQUEST_EVENTTYPES

/string/　　Sending hub name
/string/　　Sending hub territory

NXS_SYS_EVENT_NEW_EVENTTYPE_FILESを用いて、送出側にイベント・タイプを送出する。

NXS_SYS_EVENT_REQUEST_ADVERTISEMENTS
/string/ Sending hub name
/string/ Sending hub territory
NXS_SYS_EVENT_NEW_ADVERTISEMENTSを用いて、送出側ハ
NXS_SYS_EVENT_NEW_ADVERTISEMENTS
long Number of Advertisements A
(47頁)
Repeated A times:
/string/ Advertisement name
/string/ Event type name
long Priority
string Storage mode; "p" or "t" for persistent or
transient
ulong Time to live (seconds)
/string/ Originating hub name
/string/ Originating hub territory
リストにある広告に対するRemoteAdを作成する。発信ハブは、当該広告を作成したところである。システム・イベントは、送出元および当該イベントを既に見た近隣を除いて、全ての接続されている近隣に送られる。
NXS_SYS_EVENT_CONNECTION_READY
/string/ Sending hub name
/string/ Sending hub territory
送出側ハブは接続を使用する用意ができている。受信側も用意ができている場合、NXS_SYS_EVENT_CONNECTION_R
NXS_SYS_EVENT_CHANGE_ADVERTISEMENT
/string/ Advertisement name
/string/ Originating hub name
/string/ Originating hub territory
long Priority
string Storage mode; "p" or "t" for persistent or
transient
ulong Time to live
一致するRemoteAdを所与の値で更新する。このシステム・イベントは、送出元およびこのイベントを既に見たハ
NXS_SYS_EVENT_DELETE_ADVERTISEMENT
long Number of Advertisements A
Repeated A times:
/string/ Advertisement name
/string/ Originating hub name
/string/ Originating hub territory
(48頁)付随するRemoteAdsおよび関連するデータ構造を削除する。広告が削除されるときに、発信ハブによって送出される。このシステム・イベントは、送出元およびこのイベントを既に見たハブを除いて、全ての近隣に転送される。
NXS_SYS_EVENT_REQUEST_ROUTES
/string/ Sending hub name
/string/ Sending hub territory
NXS_SYS_EVENT_NEW_ROUTESを用いて、全てのルーティン

ブに全ての広告を送り出す。

EADYを用いて応答する。接続が既に確立している場合、このシステム・イベントは無視される。
NXS_SYS_EVENT_FORWARD_ADVERTISEMENT
string Advertisement name
指名された広告を全ての接続されている近隣に送出する。広告を作成したときは、それ自体にのみ送出する。

ブを除いて、全ての近隣に送られる。

グ情報を送出元に送り出す。

NXS_SYS_EVENT_NEW_ROUTES

long	Number of routes R
	Repeated R times:
/string/	Advertisement name
/string/	Originating hub name
/string/	Originating hub territory
long	Cost

送出元から所与の広告までの所与のコストでの経路の存在を記録する。AdSourceオブジェクトが作成され、これが当該広告への最初の経路である場合、ローカル・クライアントをチェックし、予約との一致を調べる。この結果、NXS_SYS_EVENT_NEW_ROUTESの送出側へのNXS_SYS_EVENT_NEW_SUBSCRIPTIONSが得られる。各経路は、転送について、別個に検討される。経路は、送出元、発信ハブ、または当該イベントを既に見たハブには転送されない。また、ルートがRemoteAdに対する最初の経路である場合、または2番目の経路であり目的地のハブが現経路の供給元である場合にのみ、そのルートが転送される。転送されるイベントのコスト・フィールドは、イベントを受信した接続のコストだけ増分される。このシステム・イベントは、接続設定の終了を印すこともできる。イベントが未だ完全に接続されていない近隣から来た場合、NXS_SYS_EVENT_CONNECTION_READYがこの近隣に送出される。

NXS_SYS_EVENT_CHANGE_ROUTES

long	Number of route changes R
/string/	Originating hub name
/string/	Originating hub territory
	Repeated R times:
/string/	Advertisement name
long	Cost

経路のコストを変更する。一致するRemoteAdおよびAdsourceを発見し、そのコスト値を更新する。このシステム・イベントは、送出元およびこのイベントを既に見た近隣を除いて、全ての近隣ハブに送られる。

NXS_SYS_EVENT_DELETE_ROUTES

long	Number of routes R
	Repeated R times:

(49頁)

/string/	Advertisement name
/string/	Originating hub name
/string/	Originating hub territory

所与の広告(RemoteAd)に対する経路(Adsource)を削除する。この経路によって供給されたあらゆる予約は失われる。この結果、NXS_SYS_EVENT_FAIL_SUBSCRITIONSが、広告に対して予約を登録してある近隣に送られる。削除された各経路の転送は別個に検討されるが、広告への最後の経路が除去されたとき場合のみである。このシステム・イベントは、送出元および発信ハブには送り出されない。

NXS_SYS_EVENT_CHANGE_CONNECTION

/string/	Sending hub name
/string/	Sending hub territory
long	Delta to sending hub's connection cost

それらの接続に対するコストを更新する。

NXS_SYS_EVENT_NEW_SUBSCRIPTIONS

long	Number of subscriptions S
	Repeated S times:
/string/	Advertisement name
/string/	Advertisement hub name
/string/	Advertisement hub territory
/string/	Filter expression
luong	Owner id in sender
ulong	Subscription id in sender

所与の広告に対して、送出元からの予約を登録する。送信側近隣上に予約を作成し、それについてRemoteAdに伝える。RemoteAdは、Neighborに対してSinkを作成する現Adsourceに伝える（未だ存在しない場合）。Sinkは、その予約に対するSubscriptionRefを追加する。広告が他のハブからである場合、新しい予約に対するシステム・

NXS_SYS_EVENT_CANCEL_SUBSCRIPTIONS

long	Number of subscriptions S
	Repeated S times:
ulong	Owner id in sender
ulong	Subscription id in sender

（50頁）リスト上の予約を取り消す。このイベントは、それ自体または近隣から見ることができる。全ての現AdSourceからの予約への全ての参照を除去する。これによって、Sinkを所有するSubscriptionRefによる予約への参照を全て削除する。取り消しは、予約を供給した可能性がある全ての近隣に送り出される。

NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS

long	Number of failures S
	Repeated S times:
ulong	Owner id in receiver
ulong	Subscription id in receiver
/string/	Hub name where failure occurred
/string/	Territory where failure occurred
/string/	Failure reason

AdSourceからの予約への参照を除去する。予約自体は、不履行による影響を受けない。予約が近隣からである場合、不履行は発信ハブに戻される。

NXS_SYS_EVENT_FORWARD_SUBSCRIPTIONS

ulong	Client id
ulong	Subscription id

新たな予約を、それを供給可能な近隣に送り出す。全ての一致するRemoteAdにこの予約について伝える。各RemoteAdは、クライアントに対してSinkを作成した、その現AdSourceに伝える（まだ存在していない場合）。Sinkはこの予約に対するSubscriptionRefを追加する。ローカルでない各RemoteAdに対して、現ソースにNXS_SYS_EVENT_NEW_SUBSCRIPTIONSを送出する。

NXS_SYS_EVENT_DELETE_EVENTTYPE

string	Event type name

イベント・タイプを破壊する。このシステム・イベントは、送出元およびこのイベントを既に見たハブを除いて、全ての接続されている近隣に転送される。

【図面の簡単な説明】

【図1】本発明の好適実施例によるネットワーク型コンピュータ・システムのブロック図。

【図2】図1のハブ上の、パブリッシャのようなアプリケーションをインストールするために実行するステップを示すフロー・チャート。

【図3】図1のパブリッシャがイベントを公表する際に実行するステップを示すフロー・チャート。

【図4】図1のサブスクライバがイベントに予約する際に実行するステップを示すフロー・チャート。

【図5】図1のハブの詳細を示すブロック図。

【図6】図5のハブのメモリに格納されるデータ構造を示す図。

【図7】図6のデータ構造の詳細を纏めた図。

【図8】ハブのクライアントについての情報を記憶する目的のために、図5のハブのメモリに格納される追加のデータ構造を示す図。

【図9】イベントのエンベロープ・データ構造のフォーマットを示す図。

【図10】イベントのルーティング・ブロックのフォーマットを示す図。

【図11】図1のハブが図6のデータ構造に移植する際に実行するステップの詳細を示すフロー・チャート。

【図12】データを移植された図6のデータ構造の例を示す図。

【図13】図1のハブが、公表されたイベントをサブスクライバに送出する際に実行するステップの詳細を示すフロー・チャート。

【図14】図1のネットワークに組み込まれた、データ・ベース・アプリケーションのブロック図。

【図15】データ・ベースがサブスクライバであるときに実行されるステップを示すフロー・チャート。

【図16】データ・ベースがパブリッシャであるときに実行されるステップを示すフロー・チャート。

【符号の説明】

100　ネットワーク型コンピュータ・システム

102，110，116　パブリッシャ

104，112，118　サブスクライバ

106，108，114　ハブ
120　ネットワーク
130，140　テリトリ
506，508　イベント・キュー
510　クライアント管理ブロック
512　イベント管理ブロック
514　プリプロセッサ・ブロック
516　格納ブロック
518　マッチ・ブロック
520　遠隔管理ブロック

【図1】

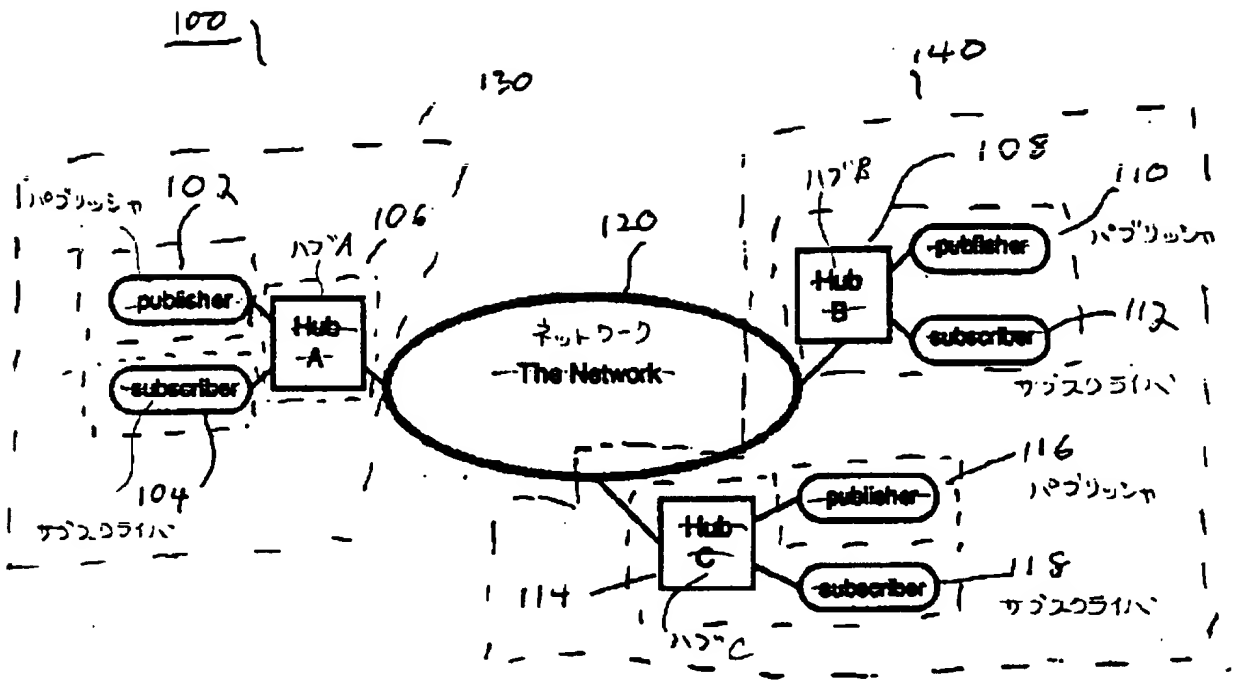


Fig. 1

図1

【図2】

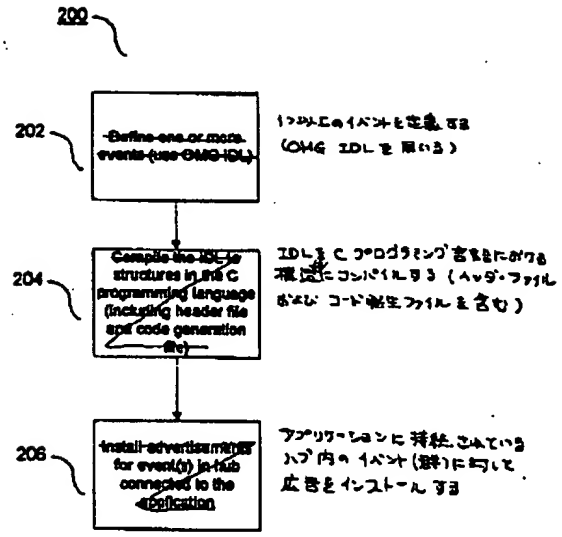


Fig. 2
Installing an Application (Publisher) on a Hub

ハブ上でのアプリケーション(パブリッシャ)のインストール
図2

【図7】

The following gives the important data fields of the classes in the above diagram.

Client: name, id, event queue, event queue postconsumer, Subscription list, publication list

Neighbor: name, id, event queue, event queue postconsumer, AdSourceRef list, Subscription list, my cost, their cost

AdSourceRef: pointer to AdSource

RemoteAd: name, eventype name, priority, storage mode, time to live, originating hub, originating territory, AdSource list, current AdSource

AdSource: Sink list, cost

Sink: SubscriptionRef list

SubscriptionRef: pointer to Subscription

Subscription: eventype name, filter expression, owner id, subscription id, reference count, (for remote subscriptions: neighbor owner id, neighbor subscription id, ad name, ad originating hub)

Fig 7
図7

【図15】

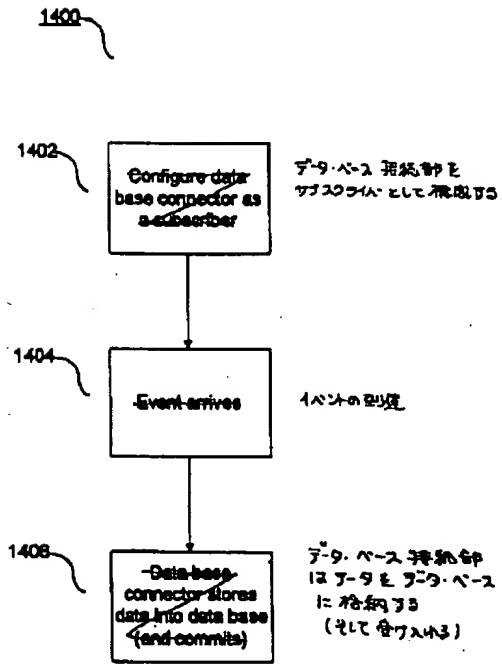


Fig. 15
図15

【図9】

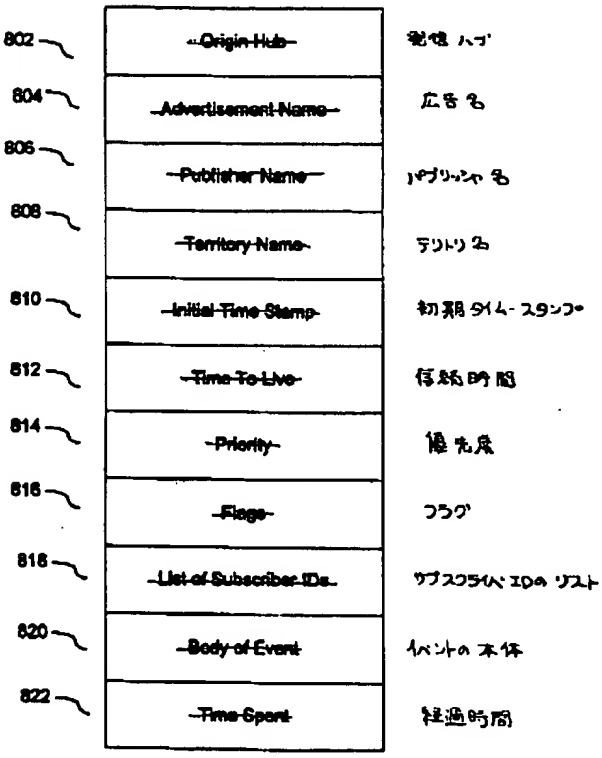
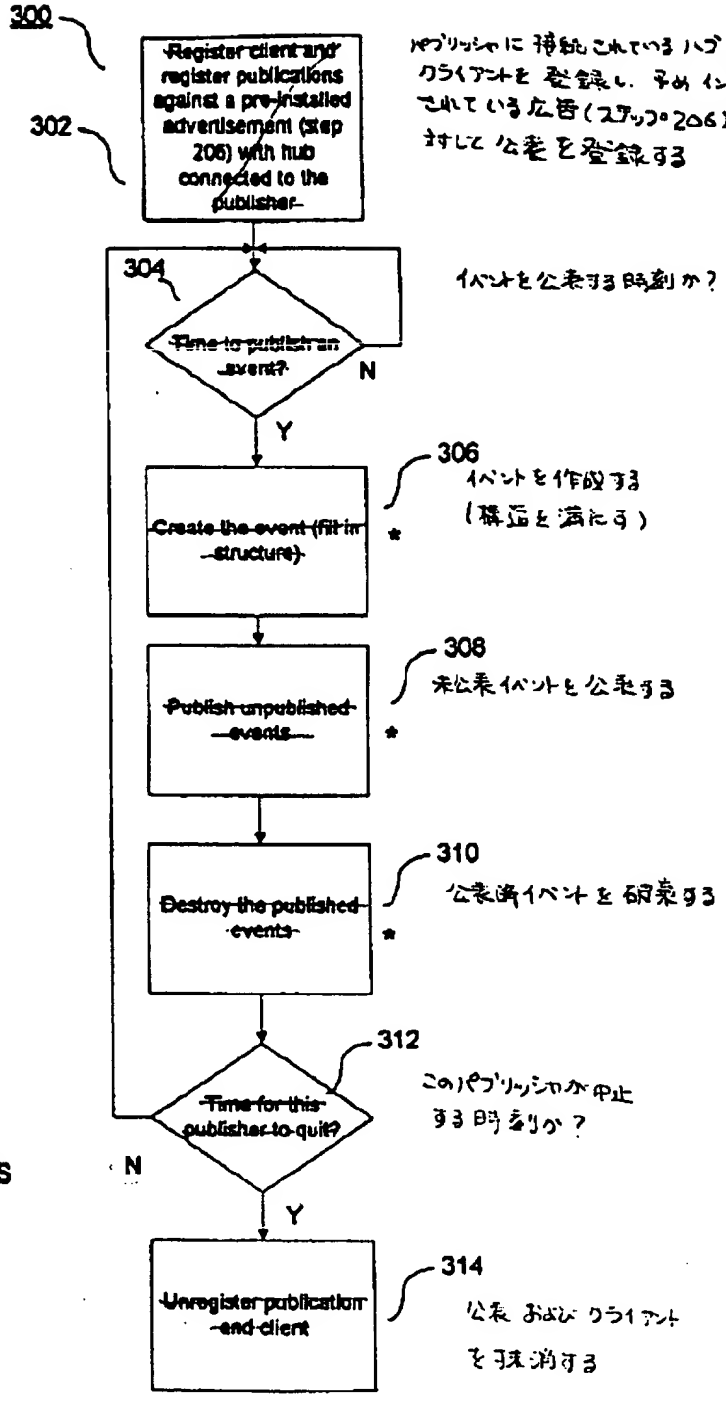


Fig. 9
図9

Envelope Format of Event

イベントのエンベロープ・フォーマット

【図3】



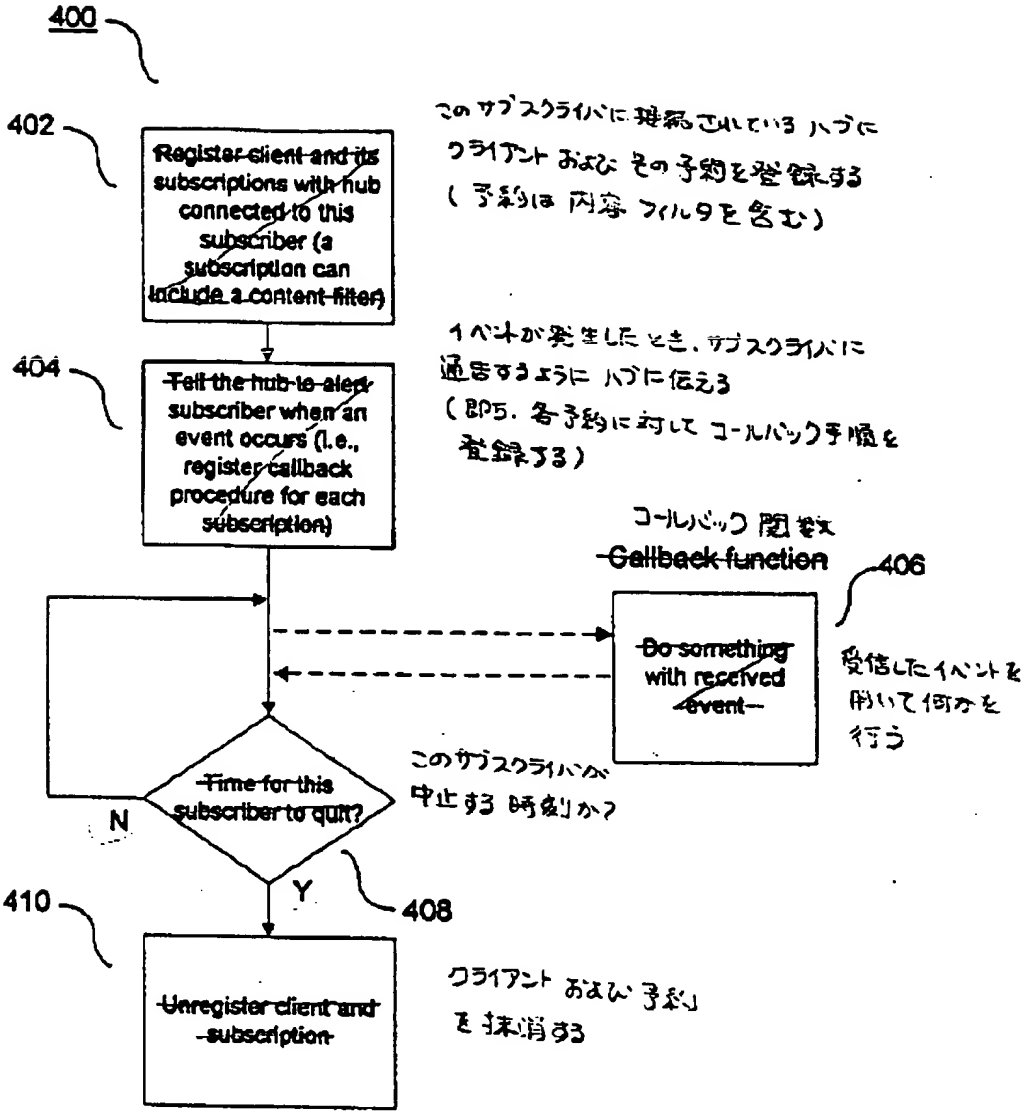
Publishing Events

Fig. 3

イベントの公表

図3

【図4】



イベントへの予約

Subscribing to Events

Fig. 4

図4

【図5】

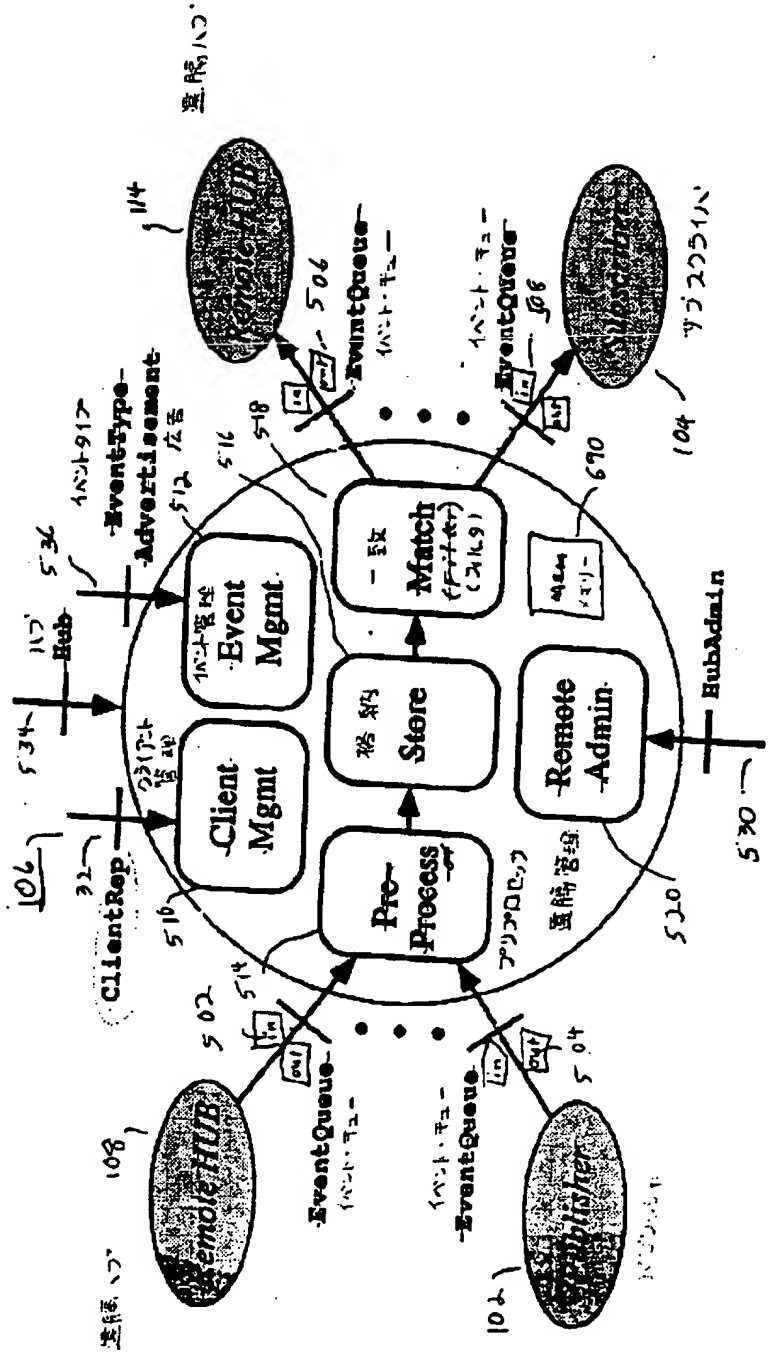

Event Mgmt
Client Mgmt
Store
Pre Process
Match
Remote Admin
Hub
ClientRep
HubAdmin
EventQueue
EventType
Advertisement
Remote HUB
Subscriber

Fig. 5　図5

【図6】

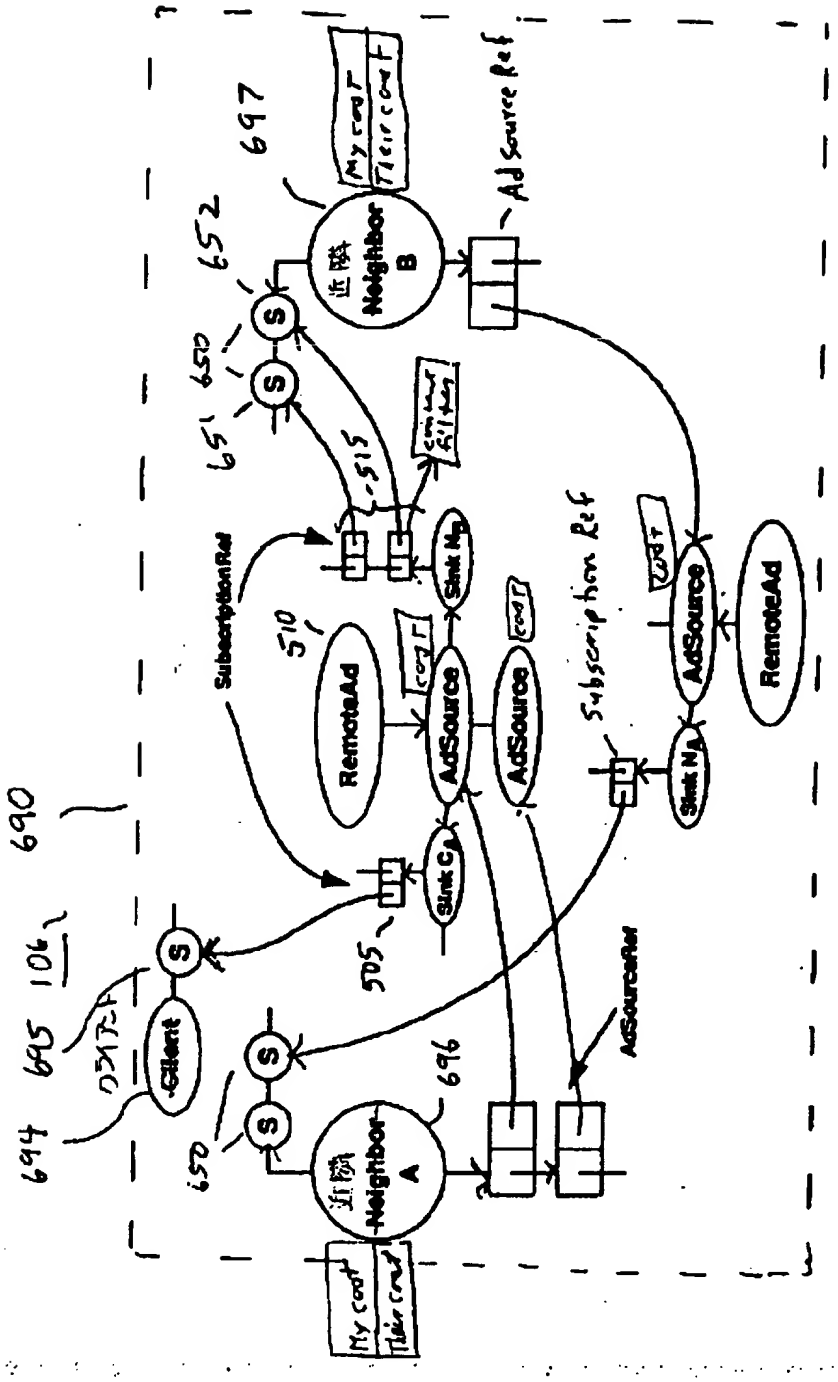

690
694
695
696
697
104
650
651
652
505
510
515
Client
Neighbor A
Neighbor B
近隣
RemoteAd
AdSource
Sink NA
Sink NB
Sink GA
SubscriptionRef
AdSourceRef
Subscription Ref
Ad Source Ref
My coord
Their coord
coord

図6

【図8】

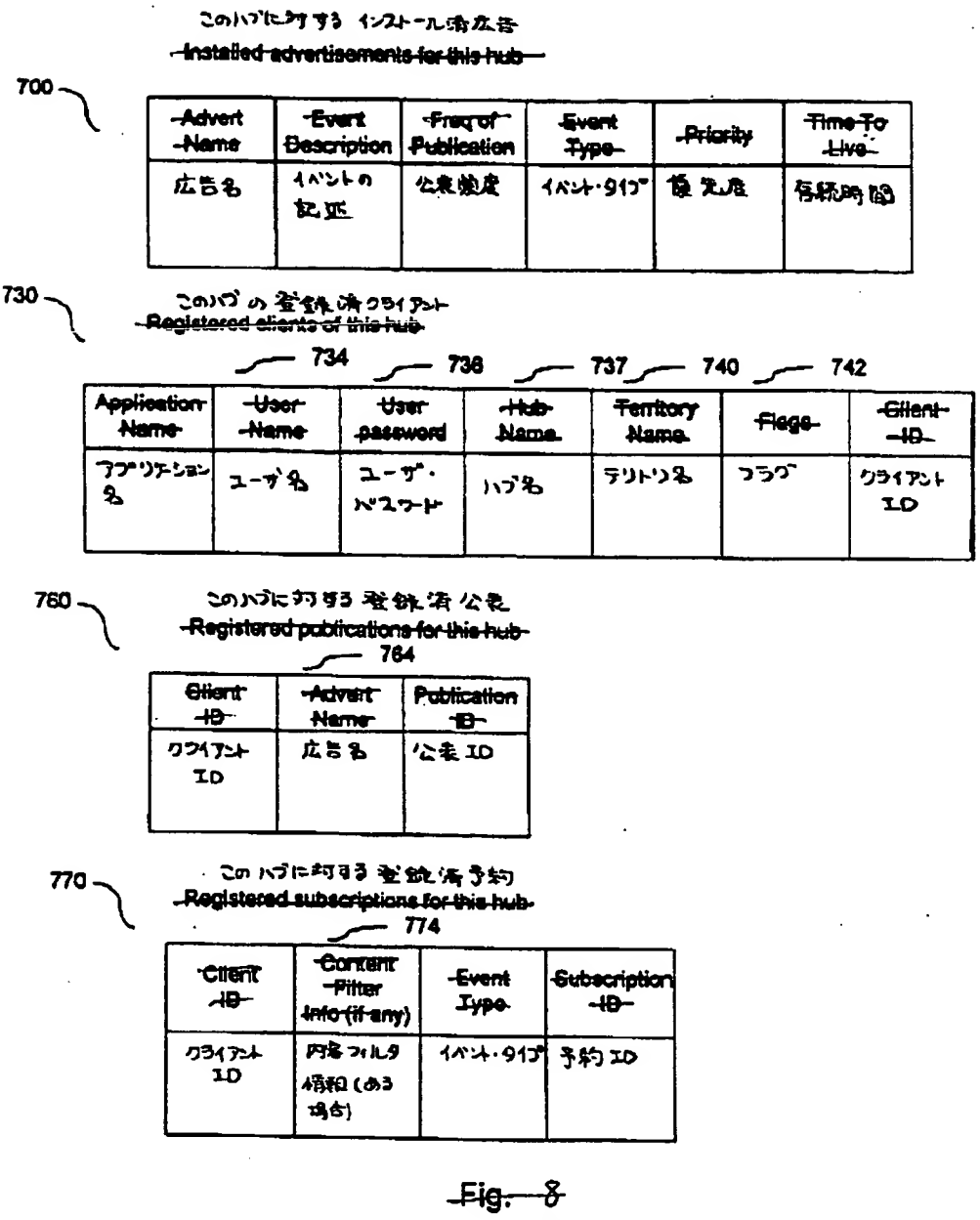

このハブに対するインストール済広告
Installed advertisements for this hub
700
Advert Name
Event Description
Freq of Publication
Event Type
Priority
Time To Live
広告名
イベントの記述
公表頻度
イベント・タイプ
優先度
有効時間
730
このハブの登録済クライアント
Registered clients of this hub
734
736
737
740
742
Application Name
User Name
User password
Hub Name
Territory Name
Flags
Client ID
アプリケーション名
ユーザ名
ユーザ・パスワード
ハブ名
テリトリ名
フラグ
クライアントID
760
このハブに対する登録済公表
Registered publications for this hub
764
Client ID
Advert Name
Publication ID
クライアントID
広告名
公表ID
770
このハブに対する登録済予約
Registered subscriptions for this hub
774
Client ID
Content Filter Info (if any)
Event Type
Subscription ID
クライアントID
内容フィルタ情報（ある場合）
イベント・タイプ
予約ID

Fig. 8

図 8

【図10】

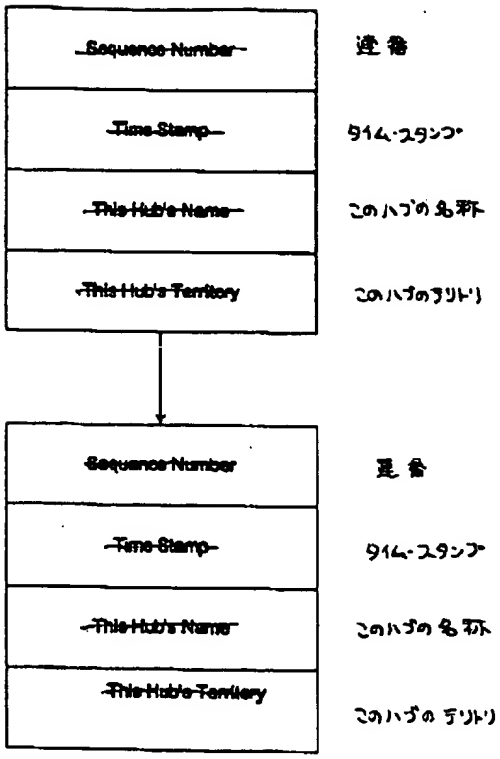


Fig. 10　図10

Routing Blocks of an Event　イベントのルーティング・ブロック

【図14】

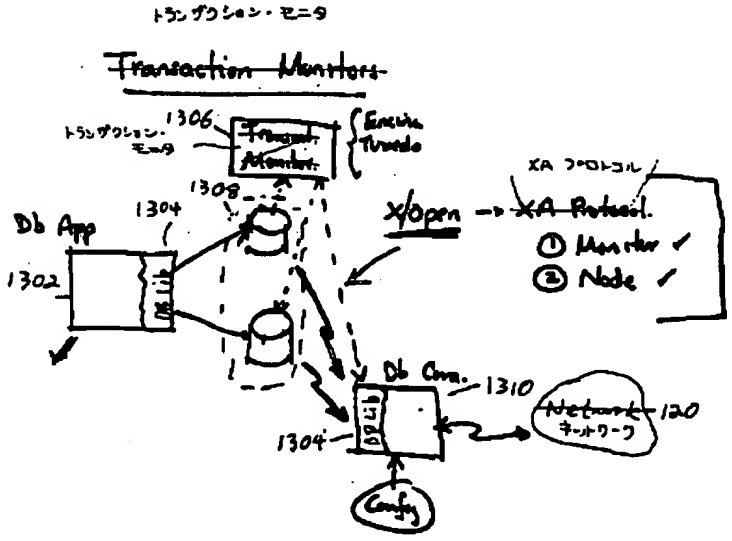


Fig. 14　図14

【図11】

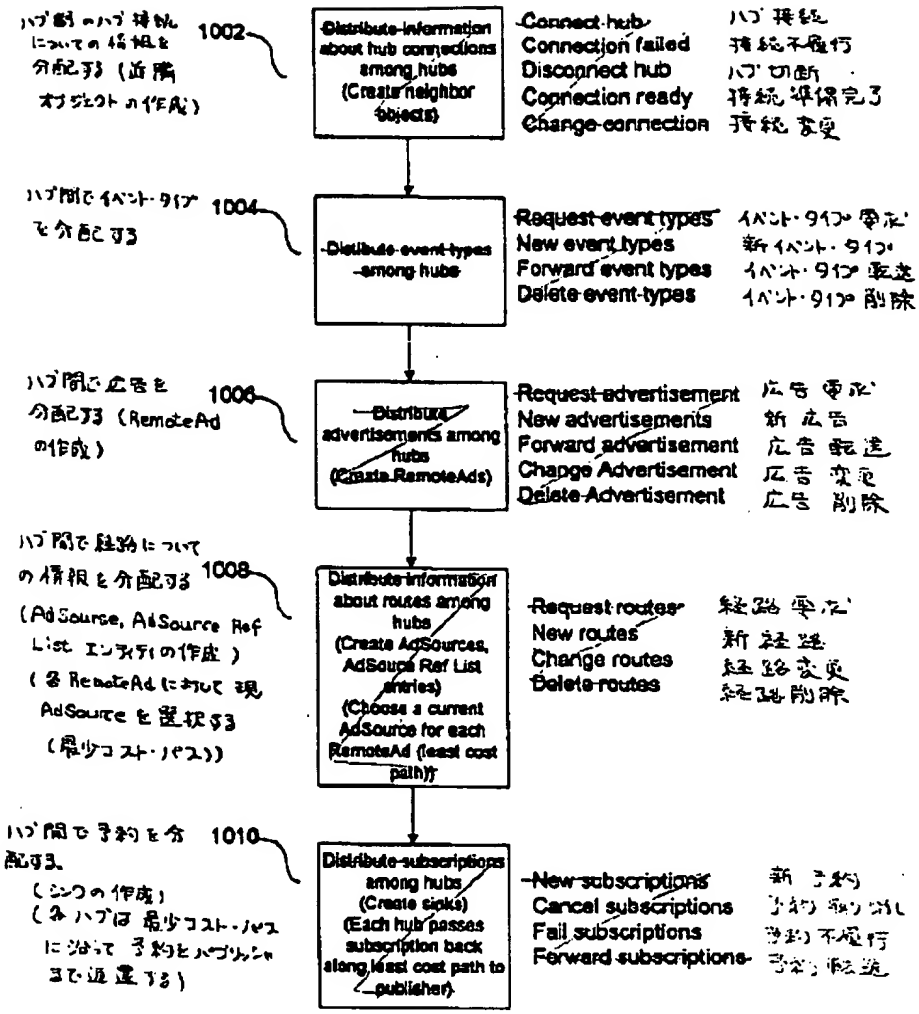


Fig. 11
Populating Data Structures of Hubs

図11

ハブのデータ構造の移植

【図12】

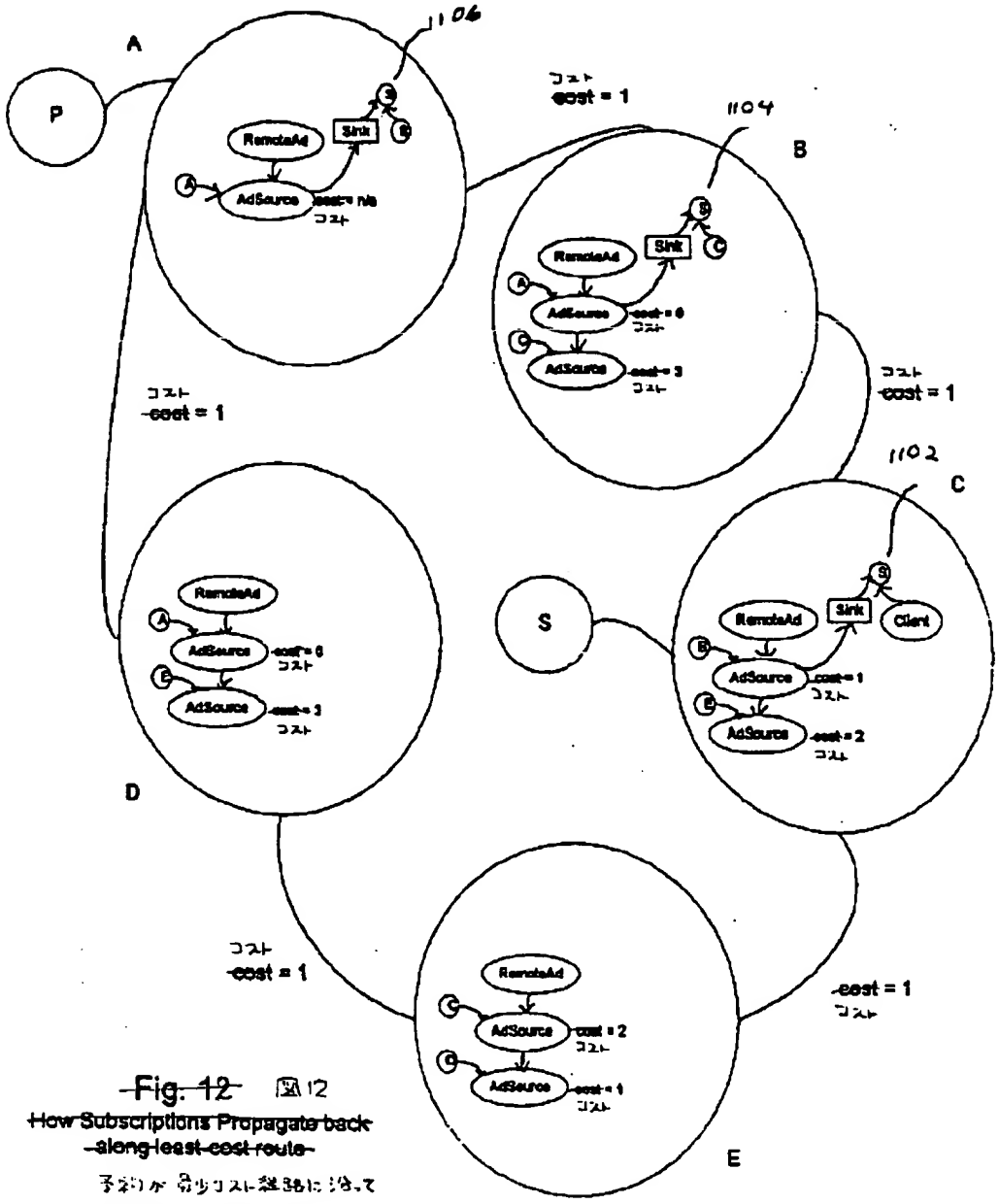


Fig. 12　図12

How Subscriptions Propagate back along least-cost route

予約が最少コスト経路に沿って逆方向に伝搬する様子

【図13】

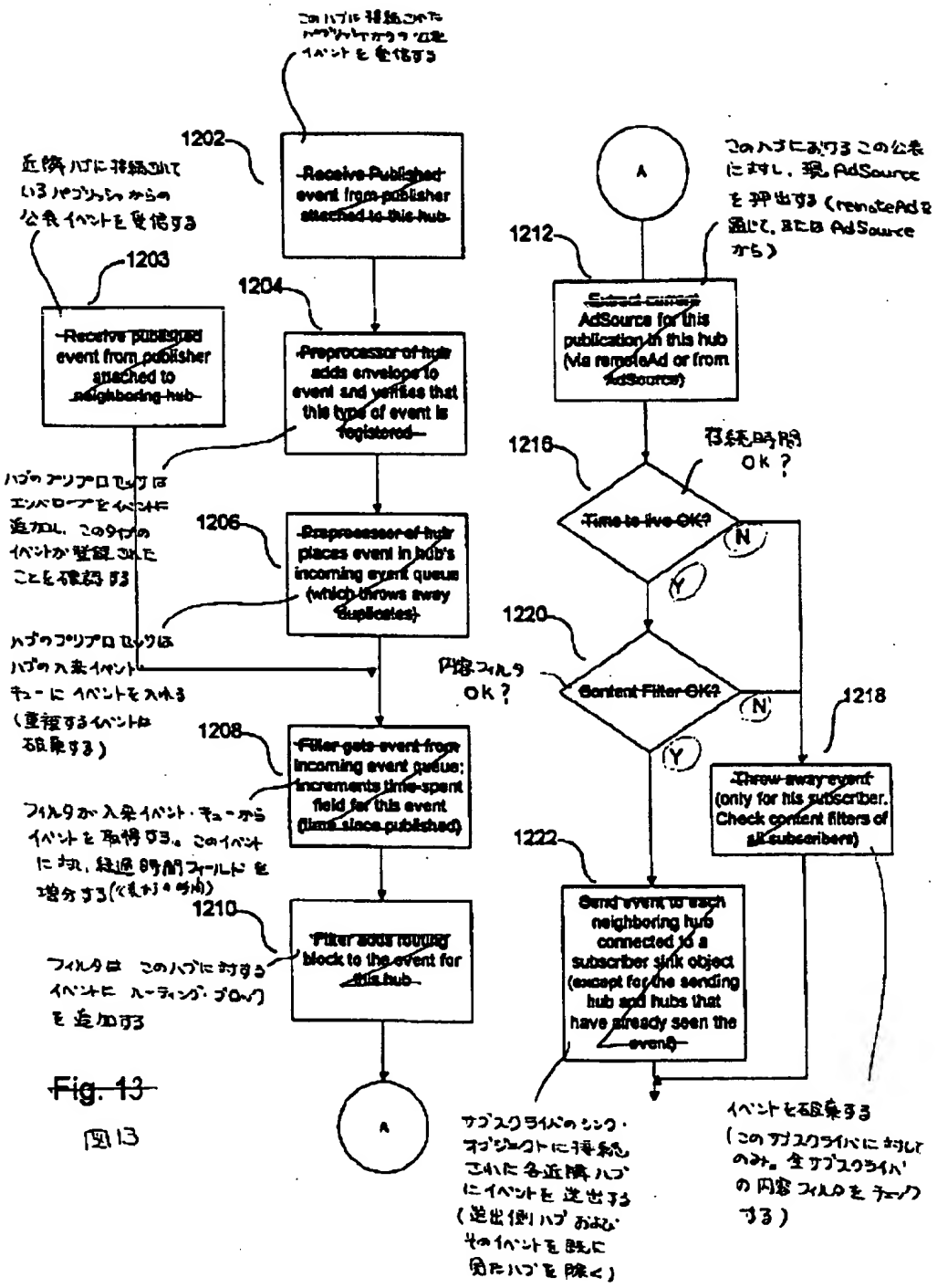


Fig. 13

図13

【図16】

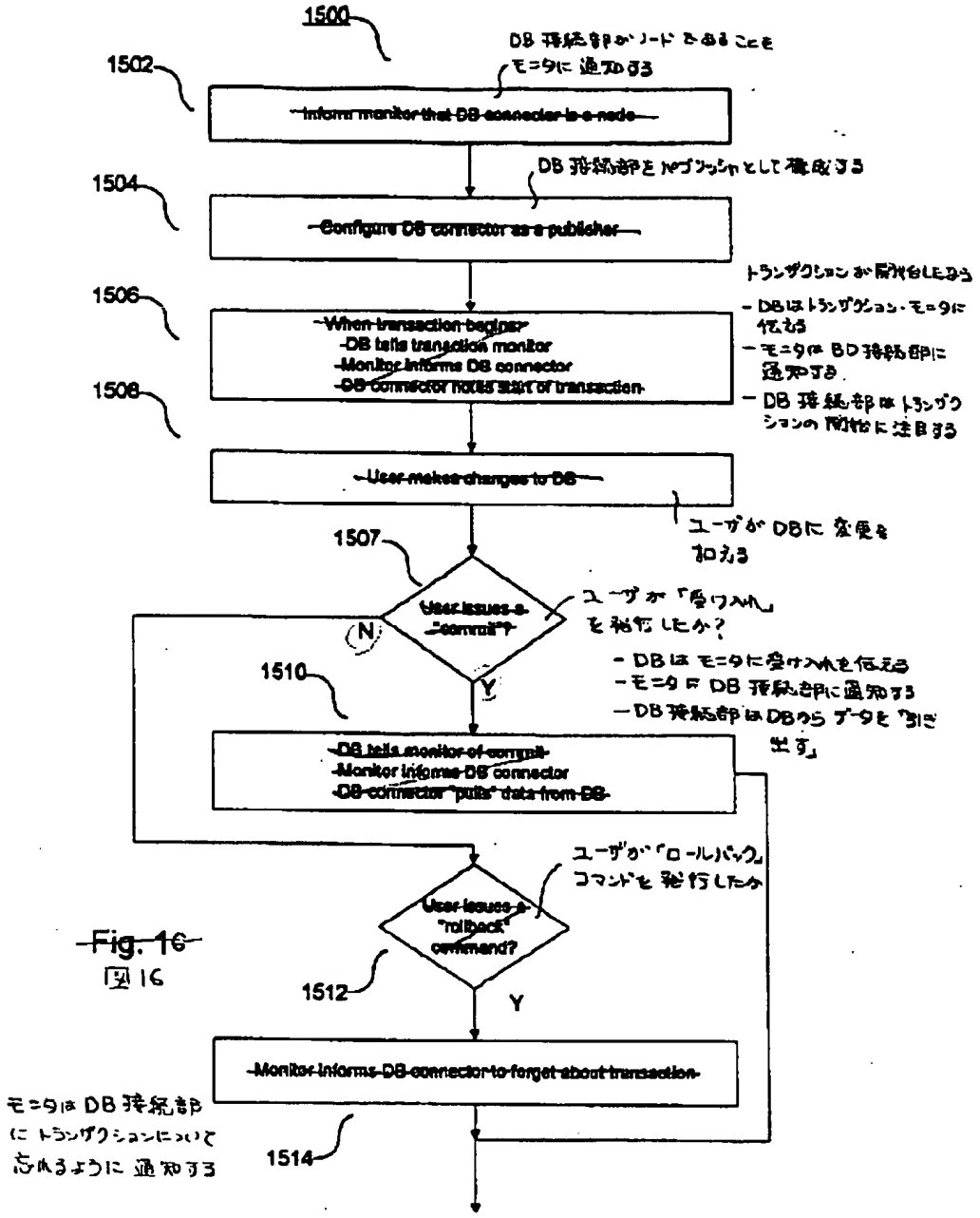


Fig. 16
図16

フロントページの続き

(71)出願人　597004720
2550 Garcia Avenue, MS PAL1-521, Mountain View, California 94043-1100, United States of America

(72)発明者　ラファエル・ブラッチョ
アメリカ合衆国カリフォルニア州95014, カッパーティーノ, シーニック・サークル 10461

(72)発明者　ティルマン・スポーカート
アメリカ合衆国カリフォルニア州95131, サン・ホセ, オヤマ・プレイス 1510

【外国語明細書】

1. Title of Invention
DATABASE NETWORK CONNECTIVITY PRODUCT
2. Claims

1. A method of connecting a data base to a publisher/subscriber network, so that the data base can publish events to the network, the method comprising the steps, performed by the data processing system, of:

providing a link so that the data base can communicate with a data base connector computer program by way of a transaction monitor computer program;

setting up the data base connector as a publisher hub in the network;

receiving input, by the data base, from a user that alters data in the data base;

informing the transaction monitor that the data has been altered;

receiving input from the user indicating that the altered data should be committed;

informing the transaction monitor that the altered data is committed; and

sending, by the data base connector, in accordance with a message from the transaction monitor that the data is committed, a published event to the network in accordance with the altered data.

2. The method of claim 1, further including the steps of:

receiving input from the user indicating that the altered data should be rolled back;

informing the transaction monitor that the altered data is rolled back; and

refraining, by the data base connector, from sending the altered data to the network in accordance with the rollback.

3. A method of connecting a data base to a publisher/subscriber network, so that the data base can subscribe to events to the network, the method comprising the steps, performed by the data processing system, of:

providing a link so that a data base connector computer program can communicate with the data base;

setting up the data base connector as a subscriber hub in the network;

receiving, by the data base connector, an event from the network;

sending, by the data base connector, data in accordance with the received event to the data base; and

committing, by the data base connector, the data in the data base.

4. The method of claim 1, further comprising the step of:

sending the event, by the data base connector, to one of its neighbor hubs, where a data structure in the data base connector indicates that the neighbor hub is on the least-cost path to a subscriber for the event.

5. The method of claim 1, further comprising the steps of:

receiving an event by the data base connector;

determining, by the data base connector, in accordance with a data structure of the data base connector, that the data base has subscribed to the event; and

sending, by the data base connector, the event to the data base.

6. The method of claim 1, further comprising the step of:

receiving an event, by the data base connector;

determining, by the data base connector, in accordance with a data structure of the data base connector, that a neighbor hub is connected either directly or indirectly on the least-cost path to a subscriber for the event; and

sending, by the data base connector, the event to its neighbor hub, so that the event can be forwarded to the subscriber.

7. The method of claim 1, wherein the sending step includes the step of:

sending an event, by the data base connector, to one of its neighbor hubs when a data structure in the data base connector indicates that the data base subscribes to events having the type of the event.

8. The method of claim 1, wherein the sending step includes the step of:

sending an event, by the data base connector, to one of its neighbor hubs when a content filter in a data structure of the data base connector indicates that the data base subscribes to events having values found in the event.

3. Detailed Description of Invention

Related Application

U.S. Application Serial No. _____, filed concurrntly herewith, entitled "Middleware For Enterprise Information Distribution" of Bracho and Jankowski.

Appendices

Appendix A, which is a part of this specification and is herein incorporated by reference, contains a summary of exemplary system events that can be passed between hubs in a preferred embodiment of the present invention.

Background of the Invention

This application relates to an information publishing system and, more particularly, to a method and apparatus for connecting a legacy data base to an information publishing system.

Certain conventional systems use a "transactional" model for exchanging information between nodes of data processing system. In a conventional transactional model, two applications initiate a synchronous "transaction" that is maintained for as long as information is being exchanged between the nodes. The transactional model requires that a transaction be defined for each business activity. When nodes on widely disparate portions of a network attempt to communicate via the transaction model, it is often difficult to define a transaction that works well on all parts of the system. Moreover, a transaction can only involve the nodes that began the transaction. Another node cannot join the transaction in the middle. This scheme is unsuited for a system in which nodes do not know about all other nodes in the system.

Many commercial enterprises still use software that was developed long ago and is no longer supported by its manufacturer. Such software is called "legacy software." In addition, many commercial enterprises use commercial software products. It is not always possible or desirable to modify existing commercial software and legacy software to operate with

a new networked system. The commercial software and legacy software may be very complex, making it difficult and expensive to modify. A company may be wary of making modifications to a stable system that is working consistently. Moreover, there may not be any computer programmers at a company who understand the legacy system in enough detail that they can make modifications to it. Lastly, a company may not have the source code for a legacy application if it purchased the application from a commercial vendor. A company also may not have a legal right to modify commercial data base software. What is needed is a way to integrate commercial and legacy data base software into a networked system without having to modify the data base software itself.

Summary of the Invention

The present invention overcomes the problems and disadvantages of the prior art by providing a "data base connector" element that can act as either a publisher or a subscriber in the network.

The present invention is implemented as a type of "middleware." Middleware is software that is located between an application program and a control-level program. Middleware is "network centric" and does not concentrate on a specific user interface or on the organization of a specific database. The described embodiment includes a plurality of "publisher" entities, who publish information, and a plurality of "subscriber" entities, who request and use the information. Publishers and subscribers are connected to each other through a network. The network is a "store and forward" network whose routing is "content-based." In a content-based routing system, information is routed based on the content of the information, and not on the addresses of publishers or subscribers in the system. In the described embodiment, information is distributed to many subscribers in parallel.

In the described embodiment, the basic quanta of information is called an "event." Publishers publish events and subscribers subscribe to events that match criteria defined by the subscriber. In the described embodiment, events are represented by data structures in the C programming language, although any appropriate representation can be used.

Publication and subscription are performed asynchronously. Publishers and subscribers do not have direct knowledge of each other. The system receives published event from a publisher and routes the event to all appropriate subscribers. Each subscriber is guaranteed to receive all events published on the system if, and only if, they match the subscription criteria specified by the subscriber.

The described embodiment includes an Application Programming Interface (API) for publishers and for subscribers. The API defines a plurality of procedures that allow respective publishers and subscribers to interface to the system. Thus, various types of publishers and subscribers can be connected to the system, as long as they use the interface procedures defined in accordance with the API. The described embodiment also includes support for a plurality of software elements such as common databases, installation and administration tools, and logging facilities. Thus, the described embodiment is termed an "enterprise system" because it can be used throughout the entirety of a commercial enterprise.

The described embodiment of the present invention makes it easy to integrate legacy systems, legacy applications, and legacy hardware into the system and attempts to minimize the amount of information that a user must learn to use the system. Because communication within the system is based on asynchronous "events," the present invention can be implemented on a heterogeneous network that includes both PCs and mainframes executing under various operating systems and running various

types of applications. The described embodiment utilizes a distributed-object environment and a preferred embodiment of the present invention implements the Common Object Request Broker (CORBA) standard.

In accordance with the purpose of the invention, as embodied and broadly described herein, the invention relates to a method of connecting a data base to a publisher/subscriber network, so that the data base can publish events to the network, the method comprising the steps, performed by the data processing system, of: providing a link so that the data base can communicate with a data base connector computer program by way of a transaction monitor computer program; setting up the data base connector as a publisher hub in the network; receiving input, by the data base, from a user that alters data in the data base; informing the transaction monitor that the data has been altered; receiving input from the user indicating that the altered data should be committed; informing the transaction monitor that the altered data is committed; and sending, by the data base connector, in accordance with a message from the transaction monitor that the data is committed, a published event to the network in accordance with the altered data.

Objects and advantages of the invention will be set forth in part in the description which follows and in part will be obvious from the description or may be learned by practice of the invention. The objects and advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims and equivalents.

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments of the invention and, together with the description, serve to explain the principles of the invention.

Detailed Description of the Preferred Embodiments

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

I. Overview

Fig. 1 is a block diagram of a networked computer system 100 in accordance with a preferred embodiment of the present invention. Networked computer system 100 includes a plurality of publishers 102, 110, and 116 and a plurality of subscribers 104, 112, and 118. Publisher 102 and subscriber 104 are connected to a network 120 through a hub 106. Publisher 110 and subscriber 112 are connected to network 120 through a hub 108. Publisher 116 and subscriber 118 are connected to network 120 through a hub 114. Hub 106 and its connected publishers and subscribers are in a first territory 130. Hubs 108 and 114 and their

connected publishers and subscribers are in a second territory 140. Other territories (not shown) also exist in network 120. "Hubs," "publishers," "subscribers," and "territories" are discussed below in turn.

As indicated by dotted lines in Fig. 1, each publisher, subscriber, and hub can be located in separate computers (102, 104, and 106), in the same computer (108, 110, and 112) or in some combination of separate computers and the same computer (114, 116, and 118). Hubs can have zero or more publishers and zero or more subscribers. Hubs can also be directly connected to other hubs.

It will be understood by persons of ordinary skill in the art that each computer (represented in Fig. 1 by dotted lines) includes a CPU, a memory, and input/output lines. These elements are not shown in the figure for the sake of clarity of example. Each computer can also include numerous other elements, such as disk drives, keyboards, display devices, network connections, additional memory, additional CPUs, etc. The publishers, subscribers, and hubs preferably are implemented as software instructions stored in a memory and executed by a processor of a computer system. Usually, the publisher, subscriber, and hub software is executed by a processor of the computer in which the software is stored, although a processor of another computer system could also execute the software.

A preferred embodiment of the invention runs under the Solaris operating system, Version 2.4. Solaris is a trademark of a registered trademark of Sun Microsystems, Inc. in the United States and other countries and is based on the Unix operating system. Unix is a registered trademark in the United States and other countries, exclusively licensed through X/OPEN, Ltd.. Networked computer system 100 uses a network communication protocol, such as TCP/IP, although any appropriate protocol can be used to implement the present invention

In the described embodiment, a "publisher" publishes events of certain types on the network 120 and a "subscriber" subscribes to events

of certain types. The publisher and subscriber operate asynchronously and are unaware of each other's existence. Use of a "publish and subscribe" model insulates applications from the network and from each other. Publishing can be thought of as broadcasting an event. The event type is analogous to a radio station. Applications interested in a particular station subscribe (or tune in to) a specific event type. Just as in radio broadcasting, where all parties involved must agree in advance on a set of possible frequencies, applications in the described embodiment must agree in advance on a predetermined set of event types.

The described embodiment of the present invention uses "subscription by content." Subscribers specify what they want based on an event type and on the content of the event. The described embodiment makes use of "store and forward" techniques for events. This term means that the system buffers event, so that, even if a publisher is off-line, a subscriber can still retrieve events. Similarly, a publisher can publish events even if a subscriber is off-line.

The described embodiment is managed in units of hubs. A hub, such as hub 106, is a local connection point for a publisher and/or a subscriber. Both publishers and subscribers are called "clients" of hubs. A publisher uses an "advertisement" to tell the system what types of events it intends to publish and how it intends to publish them (e.g., daily, as each event trigger occurs, etc). Advertisements are registered with the hub connected to the publisher during the installation of the publisher. An advertisement must have a unique name. In addition to containing a name of an event type to be published, an advertisement contains other information about the publisher. This information can include the priority level for the published events, for how long the events are valid, etc. Hubs transmit advertisements to all potential subscribers. Hubs transmit published events to all subscribers who have subscribed to events of that type, whose content matches the subscriber's subscription.

II. Publishers and Subscribers

The following paragraphs describe the steps required to create an application (either a publisher or a subscriber) and to install it on a hub. The following paragraphs further describe the steps performed by an example publisher to publish events and by an example subscriber to subscribe to events. The described embodiment allows a programmer to write software for applications (publishers and subscribers) in a high level language and to interface with the hub using routines defined in an API and using event manipulation routines generated for each user-defined event type.

Fig. 2 is a flow chart showing steps performed to install an application, e.g., a publisher, on a hub. Fig. 3 is a flow chart showing steps performed by an installed publisher to publish an event. Fig. 4 is a flow chart showing steps performed by a subscriber of Fig. 1 to subscribe to events of a certain type and content. As discussed above, both publishers and subscribers preferably are implemented as software programs executed by a processor.

The described embodiment is centered around the sending (publication) and receiving (subscribing) of events. Before a publisher can publish events, the publisher must define and advertise the events that it will publish. In order for the events to make sense, publishers and subscribers need to understand each other. For this reason, the described embodiment uses a standard specification language to define events. In step 202 of Fig. 2, the publisher defines one or more events that can be published by that publisher. The described embodiment of the described embodiment allows definition of events using an industry standard Interface Definition Language (IDL). IDL is a standard interface propagated by the Object Management Group (OMG), although any applicable syntax for describing the structure of the event could be used

in conjunction with the described embodiment. In step 204, the OMG IDL definitions of the events are translated into data structures in the C programming language.

Assume, for example, that an application publishes "SalesOrder" events. Before the application could be added to the system, and assuming that no other applications dealing with SalesOrder events already existed, a programmer would define a "SalesOrder" event using OMG IDL as follows:

```
//filename: order:ddl

interface SalesOrder {
        struct Address {
                string street;
                string city;
                string state;
                unsigned long zip;
        };

attribute string customer_name;
attribute Address customer_address;

};
```

The above syntax defines an event as an "interface" and defines the data members in the event as "attributes."

After one or more events have been defined, the events are translated into structure definitions of the C programming language. Preferably, this translation is performed by an OMG IDL compiler, which generates a header file and a code file for each event. The generated header file contains simple C-language structures that represent the event declarations for the typed functions defined in the code file. The C structure definition generated for the above example event of type

SalesOrder is:

```
typedef struct SalesOrder_Address {
        nxsString street;
        nxsString city;
        nxsString state;
        nxsULong zip;
} SalesOrder_Address;

typedef struct SalesOrder {
        nxsString customer_name;
        SalesOrder_address customer_address;
} SalesOrder;
```

A person of ordinary skill in the art will easily understand how the IDL of the example translates into the C programming language structure definitions shown above.

The code file generated from the event definition contains procedures for manipulating events of the defined type (here, for the SalesOrder event). The procedures perform at least the following operations:

For use by a publisher:

Create event of type SalesOrder,

Publish event of type SalesOrder,

Destroy event of type SalesOrder

For use by a subscriber:

Print contents of event of type SalesOrder,

Register Subscription for event of type SalesOrder,

A person of ordinary skill in the art will understand that, although the above plurality of procedures is generated for each defined event, the operation of the procedures differ due to the differences in the structures of the event. The procedures also differ due to the fact that the described embodiment uses typed function to prevent programming errors.

The process of installing a process application in a hub includes storing one or more advertisements in the hub, as shown in step 206. These advertisements define types of events that the hub knows about. An advertisement includes the name of the advertisement, a description of the event that a publisher publishes (e.g., a SalesOrder event), a description of how the publisher publishes its events (e.g., daily, when an event trigger occurs, etc.), the name of the event type, the priority level of the events, and the expiration time on the events (also called "time-to-live"). Although not shown in the Figure, the described embodiment also allows access control permissions for the various event types to be set.

The described embodiment also includes predefined routines in an API, which are available to the publisher and subscriber software and which include routines to perform at least the following functions:

Register a client,

Reestablish a client with an existing session,

Unregister a client,

Registering intent to publish,

Unregistering a publication,

Publishing an event,

Subscribing to an event,

Reactivating an active subscription to an event, and

Cancelling a subscription,

Other APIs in accordance with the present invention may include somewhat different API functions. In addition, the API preferably includes a "can_subscribe" routine and a "can_publish" routine that return a Boolean value indicating whether the calling application can subscribe to or publish events of a certain type. The API also includes a "get_current_envelope" routine that returns information about the publisher of an event and how the event was published. (This routine can only be called from the call-back procedure). A call-back procedure is called, e.g., when a subscriber receives an event. Fig. 9 shows a format of an envelope data structure for an event, which is discussed below in connection with event routing.

It will be understood that when the event description for a publisher or a subscriber is initially translated into, e.g., C, the event header file, whose generation is discussed above, is included within the application software source code. Thus, in the example, the publisher has access to the definition of the SalesOrder event. The translation also links the event manipulation routines and API routines into the application at this time. The hub, along with other hubs, transmits a notice of the

advertisement existence to potential subscribers, as discussed below.

Once a publisher is installed in a hub, it is free to publish advertisements and events. Fig. 3 is a flow chart showing steps performed by a publisher of Fig. 1 during operation of the publisher. Steps involving a call to the event manipulation routines that are created by compiling the IDL are indicated by an asterisk in the Figure.

As shown in step 302 of Fig. 3, the publisher first registers itself as a client with the hub to which it is connected. The publisher next advises the hub of which "advertisement" it will use when publishing. An advertisement represents an "intent to publish" and tells the hub what type of event the publisher will be publishing. To advise the hub in step 302, the publisher tells the hub the name of an advertisement that has been previously installed in the hub (see Fig. 2).

In step 304, the publisher waits until it is time to publish an event. Some publishers, such as in the example above, publish each event at the time that it occurs. Other publishers wait until a specified time, such as once a day, and publish all unpublished events at that time. For example, a first publisher may publish an event whenever a human operator enters a sales order into the system. A second publisher may publish the same type of event on an hourly basis. Once the publisher determines that an event is to be created, it calls (in step 306) one of the event manipulation routines that creates a C structure having the proper format for the event to be published.

In step 308, the publisher calls one of the event manipulation routines to publish unpublished events as discussed below in more detail. In step 310, the publisher calls one of the event manipulation routines to destroy the published event. (In general, in the described embodiment, whatever entity is responsible for allocating an event is also responsible for destroying that event.)

In step 312, the publisher determines if it should cease execution. If not, control returns to step 304. If so, in step 314, the publisher unregisters itself with the hub and ceases execution. Advertisements are kept in the hub to indicate that a publisher could execute and send those events. This information is needed to build routes to subscribers, as described below. Although not shown in the Figure, other publishers connected to the same hub may operate asynchronously with the steps of Fig. 3.

Subscribers subscribe to events of particular event types. Fig. 4 is a flow chart showing steps performed by a subscriber of Fig. 1 during operation of the subscriber. The subscriber first registers itself as a client of the hub. Then for each subscription, it registers a "call-back" procedure.

As shown in step 402, the subscriber next registers itself as a client with the hub to which it is connected. The subscriber then registers a "subscription" for the event type that it wishes to receive through the hub. (The subscriber can look at the hub to see what types of events are

advertised.) A subscription specifies a type of event, and can further specify a "filter" indicating that the subscriber wishes to receive only events of a certain type that also have certain values in certain fields.

In step 404, for each subscription, the publisher defines a predetermined "call-back" procedure. The call-back procedure will be initiated by the hub whenever the hub determines that the subscriber has received an event of a certain type. The format of the callback may be different for each operating system on which the present invention is implemented and the call-back procedure can be practically any procedure that is performed when an event is received. For example, in step 406 of Fig. 4, the call-back procedure could print the contents of the event (using one of the event manipulation procedures designed to print an event of that type). Because a call-back procedure is defined for each subscription, a call-back procedure must be defined for each type of event to which the subscriber plans to subscribe. In the described embodiment, an event passed into a call-back procedure is destroyed when the call-back procedure returns. Thus, if the subscriber wants to keep any information of the event, the information should be copied out of the event within the call-back procedure.

The subscriber loops until it is time to cease execution (see step 408). The call-back procedure may be activated zero, one, or many times during this loop, as events of the specified type and values are received by the hub. If an event subscribed to by the subscriber is received, the

call-back procedures is executed in step 406. If, in step 408, the subscriber determines that it is time for it to quit, the subscriber optionally unregisters its subscription, unregisters itself with the hub, and ceases execution. Subscriptions can also be retained in the hub, to receive future events while disconnected. The hub will queue events arriving for the subscriptions. Although not shown in the Figure, other subscribers connected to the same hub may perform steps asynchronously with the steps of Fig. 4.

The following examples are written in the C programming language and show example software for a publisher and subscriber.

```
/*
 * publish.c
 */

#include <nexus.h>
#include "nxs_output/order_nxs.h"

int main()
{
        SalesOrder *event;
        nxsClientID id;
        nxsPublicationID pub_id;


        /* Register the Client and the publication */
        nxsCreateClient ("order publisher", NULL,NULL, 0, &id);
        nxsRegisterPublication (id, "order_advertisement", &pub_id);


        /* Create the event */
        event = nxsCreate_SalesOrder();
        event->customer_name = "Jim Nexus";
        event->customer_address.street = "12345 Anistreet";
        event->customer_address.city = "Mountain View";
        event->customer_address.state = "CA";
```

```
        event->customer_address.zip = "95000";


        /* Publish Event */
        nxsPublish_SalesOrder (id,pub_id,event,0,0);


        /* Clean up */
        nxsDestroy_SalesOrder(event);


        /* Unregister the publication and client */
        nxsUnregisterPublication (id,pub_id);
        nxsDestroyClient (id);

        return (0);
}

/*
 * subscribe.c
 */

 #include <nexus.h>
 #include "nxs_output/order_nxs.h"

/* Callback function to be called when each event is received */
void event_callback (
        nxsClientID id,
        nxsSubscriptionID sub_id,
        nsxULong seq_num_high,
        nsxULong seq_num_low,
        SalesOrder *the_event,
        void *client_data)
{

        char *st;

        printf("Event received!\n");
        st = nxsStringify_SalesOrder(the_event);
        printf(st);
        free(st);

}

int main(int argc, char **argv)
```

```
{
        Sales Order *event;
        nxsClient ID id;
        nxsSubscriptionID sub_id;
        /* Register Client and subscription */
        /* (Only subscribe to specific ZIP codes) */
        nxsCreateClient ("order subscriber",NULL,NULL, 0,&id);
        nxsRegisterSubscription SalesOrder(id,
                "customer_address.zip == 95000",
                event_callback,NULL,&sub_id);

        /* Main loop */
        nxsMainLoop(id);

        /* Unregister subscription and client */
        nxsUnregister Subscription(id,sub_id);
        nxsDestroyClient(id);

        return(0);

}
```

III. Hubs

Fig. 5 is a block diagram showing details of hub 106 of Fig. 1. Hubs 108 and 114 of Fig. 1, and indeed all other hubs in the described embodiment, have a similar structure. Hub 106 is connected to remote hubs 108 and 114 via network 120 (or directly) and to zero or more publishers 102 and zero or more subscribers 104. Hub 106 also receives information 530 relating to hub administration, information 532 relating to management of its clients (publishers and subscribers), and information 536 relating to system event management.

All input and output to and from hub 106 is queued. For example, events received by hub 106 from remote hub 108 and from

publisher 102 are stored in event priority order in respective event queues 502, 504 until hub 106 has time to process them. As a further example, events to be sent by hub 106 to remote hub 114 and to subscriber 104 are stored in event priority order in respective event queues 506, 508 in memory until hub 106 has time to send them. Hubs distribute events using a first-in, first-out policy. This means that all events having a same priority level will be delivered by hub 106 in the order that they are accepted from the publisher. All events with a higher priority level are delivered earlier than waiting events with a lower priority level. The described embodiment guarantees to keep the ordering of like-events from a single publisher as the events move though the system. This is important, for example, in transaction processing where an order of updates must be maintained. Inter-publisher ordering is not guaranteed, since it depends on routing and availability issues.

Fig. 5 also shows the following functional blocks inside hub 106: client management block 510, event management block 512, preprocessor block 514, store block 516, match block 518, and remote administration block 520. Client management block 510 deals with registering and unregistering clients. Event management block 512 deals with managing system events, such as receipt of new connections, types, routings, advertisements, and subscriptions. Pre-process block 514 performs tasks such as adding envelope information to an event. Store block 516 is a memory of the hub. Match block 518 adds routing information to events,

filters events in a manner described below in connection with event routing, and routes events to appropriate other connected hubs. Remote administration block 520 performs system administration tasks.

Fig. 8 is a diagram of data structures stored in a memory 690 of hub 106 of Fig. 1 that is accessible by all the functional blocks of Fig. 5. The data structures include a table of installed advertisements 700, a table of registered clients 730, a table of registered publications 760, and a table of registered subscriptions 770. Data is placed in the data structures of Fig. 8 as the publisher and subscriber perform the various steps of Figs. 2, 3, and 4 as follows.

Step 202 of Fig. 2 fills entries of installed advertisements table 700.

Steps 302 and 402 of Figs. 3 and 4 fill entries of registered clients table 730. The user name field 734 and user password field 736 may have a NULL value to indicate that the current user ID should be used. Hub name 737 and territory name 740 can be set to NULL to indicate use of a default hub and territory. Flags 742 include, for example, a Boolean value "call-back-on-any-thread," which indicates that event call-backs come on new threads.

Step 302 of Fig. 3 also fills entries of registered publications table 760.

Advertisement name 764 is the name of an advertisement that has been installed in the hub. Step 404 of Fig. 4 also fills entries of

registered subscriptions table 770. Content filters 774 are discussed below. A name of a callback procedure is kept locally in the subscriber (by the C API). The API finds the callback from the subscription ID. A client data field, which is also kept by the API, allows information associated with the subscription to be passed back with each event callback. The information that is passed back has meaning to the subscribing application.

The following paragraphs describe the various types of content filters that can be specified by a subscriber when it registers a subscription. As shown in the above example, subscribers can request that the hub filter the incoming flow of events and only pass events to the subscribers that match certain criteria. A filter preferably is specified as a expression string sent by the subscriber as a parameter to the routine for registering subscriptions. The expression string syntax for a content filter in the described embodiment is as follows. (Of course, any appropriate syntax could be used):

symbol	meaning	types allowed for
=	equal to	all basic types
!=	not equal to	all basic types
>	greater than	numeric and string types
>=, <=	greater than or equal	numeric and string types
and	logical AND	expressions or Booleans
or	logical OR	expressions or Booleans

not logical NOT expressions or Booleans

Event attribute names are used to denote which field in the event should be compared. Sub-fields (those inside structures) are specified using dot notation. Names can also be enumerated types, as is known to persons familiar with the C programming language. Once a content filter has been specified, it is used during event routing as described below in connection with Figs. 9-13. Information describing each content filter for a subscription is stored in field 774 of Fig. 8.

Fig. 6 is a diagram showing data structures stored in a memory 690 of hub 106 of Fig. 5. All hubs contain similar data structures. The described implementation uses a distributed objects model, but any appropriate organizational structure could be used. Details of the data structures of Fig. 6 are listed in Fig. 7 . Ideally, the data structures of Fig. 6 describe all advertisements in the system and indicates a neighbor hub to which the current hub should send events of certain types. Figs. 11 and 13, respectively, describe the steps to populate the data structures of Fig. 6 and to use to the data structures of Fig. 7 to route events among the hubs. Fig. 12 shows an example of the data structures populated with data.

Hub 106 (the "current hub") includes data objects representing: each client of the current hub that is a subscriber (indicated by "Client" 684 and "S" 695) and each neighboring hub connected to the current hub ("Neighbor A" 696 and "Neighbor B" 697). The current hub keeps track

of the subscription objects of its neighbor hubs. Subscription objects "S" 650 represent all subscribers that can be reached through the respective neighbors. Each neighbor object has two associated cost values: "mycost" and "theircost". "Mycost" represents a cost of sending information from the current hub to the neighbor. "Theircost" represents a cost of sending information from the neighbor to the current hub. "Theircost" used to define a route, as described below. The subscribing hub will ask the "publishing" hub with the lowest "theircost" to forward the events. "Mycost" preferably is only used to inform the neighbors of how much it would cost to the current hub to send information through that link. A "cost" may be figured in a number of ways. For example, a cost may represent a financial cost, a cost in units of time, a cost as a measure of convenience, or any other appropriate cost.

Hub 106 contains a ("RemoteAd") object for each advertisement that has been registered in the system (i.e., for each intent to publish, step 202 of Fig. 2). Each RemoteAd object points to one or more "AdSource" objects, each of which represents a path between the current hub and the hub on which the publisher of the ad resides. Each AdSource object stores a cost associated with its path to the original publisher of the ad. Thus, the "cost" value for each AdSource contains the total cost for sending an event from its source to the neighbor of the current hub or, alternately, to the current hub.

Each AdSource object has an associated list of "sink objects."

(SinkCa and sinkNb preferably are located in a single list, although not shown that way in the Figure.) Each sink object has an associated list of subscriptions. For example, SinkCa has a list 505 of all subscriptions of Client 694 that matched to the advertisement of RemoteAd 510. Similarly, SinkNb has a list 515 of all subscriptions of Neighbor B (and of all subscriptions that can be reached through Neighbor B) that have matched the advertisement of RemoteAd 510.

Each neighbor object also has an associated AdSourceRef list that points to AdSources for all paths between a publisher and the neighbor. The paths are used for routing system events between hubs, as described below.

IV. Event Routing Between Hubs

Fig. 11 is a flow chart showing details of steps performed by hubs 106, 108, and 114 of Fig. 1 to populate the data structures of Fig. 6 . These steps are performed, for example, when the network is first initialized. Subsequently, various hubs may issue "system events" to tell other hubs that changes have occurred in hub connections, event types, advertisements, routings, subscriptions, etc. Appendix A, which is a part of this specification and is herein incorporated by reference, contains a summary of exemplary system events that can be passed between hubs in the described embodiment.

In Fig. 11, the name of the system event of Appendix A that

affects a step in the flow chart is shown next to the step. In step 1002, hubs send system events to each other to define the physical connections between hubs in the network. Each hub creates neighbor objects representing physical connections to its neighbor hubs (see Fig. 6). In step 1004, the hubs send system events to each other to define the types of events for which advertisements can be published. Each hub makes sure it knows all the events. This step is performed when adding a hub to a territory. In step 1006, hubs that are connected to a publisher send system events to other hubs, which are forwarded among the hubs, to distribute advertisements among the hub. For example, the system event NXS_SYS_EVENT_CREATE_NEW_ADVERTISEMENT of Appendix A causes the creation of RemoteAd objects in the hubs (see Fig. 6).

In step 1008, hubs that are connected to a publisher send system events to other hubs. The system events are forwarded among the hubs to determine routes from the publisher's hub to other hubs. (The system event NXS_SYS_EVENT_NEW_ROUTES causes the creation of AdSource objects in the hubs (see Fig. 6). To register the existence of a route, an initial hub sends a system event to its neighbors, containing an advertisement name/ID, its (the publishing hub's) name and territory, and a cost associated with sending an event from the publishing hub to the current hub. In the described implementation, the cost is initially set to zero. Each neighbor hub repeats this process, adding the cost of the connection over which that hub received the event. In each hub, if this is

the first route to the hub, then the local clients are checked for subscription matches (This action may result in a NEW_SUBSCRIPTION system event to the sender of NEW_ROUTES). Each route is considered separately for forwarding. The route is not forwarded to the hub from which it was received, to the originating hub, or to a hub that has already seen the event (determined from the routing list, as discussed below). The route is only forwarded to other hubs if it is the first route for the RemoteAd or if it is the second route and the destination hub is the current route provider.

In step 1010, hubs connected to a subscriber send system events to other hubs to register a subscription against an advertisement known about by the other hubs. (The system event NXS_SYS_EVENT_NEW_SUBSCRIPTIONS sends subscriptions to other hubs, see Fig. 6). A hub receiving the system event creates a subscription object (see Fig. 6) and tells the RemoteAd about it. An AdSource having a least-cost path is called the current AdSource. The RemoteAd tells the current AdSource, which creates a sink and a subscription object of the Neighbor object (if a sink and a subscription object do not exist). The sink then adds a SubscriptionRef entry (e.g., 505, 512) for the subscription object. If the advertisement was received from another hub, the current hub forwards a system event for the new subscription to the neighbor that is a part of the current least-cost route.

This forwarding of subscriptions to other hubs is an important

aspect of the present invention. It essentially creates a chain of subscriptions from a hub of a subscriber back to the hub of the publisher. This chain follows the least-cost route between the publisher and the subscriber.

Fig. 12 shows an example of the creation of a chain of subscriptions from a hub of a subscriber to a hub of a publisher. Fig. 12 shows five hubs A, B, C, D, and E. Hub A has a connected publisher P. Hub C has a connected subscriber S. In the example, each connection has a cost of "1". Thus, for example, there is a route of total cost = 2 between hub A and C and a route of total cost = 3 between hub A and C.

Fig. 12 shows the status of the hubs after step 1010 of Fig. 11. Connections, event types, and routes have been distributed among the hubs and the RemoteAd, AdSource, Neighbor, and AdSourceRef objects have been created in each hub.

In the example, subscriber S registers a subscription on hub C. (Hub C had previously created a subscription object 1102 when S registered itself as a client, indicating that the subscriber S is a client of hub C). Hub C then routes a "new subscription system event" (a "subscription") to a neighbor hub that is on a least-cost path to the publisher of the advertisement for the event. In the example, hub C will route the subscription to hub B, since the cost from hub B to the publisher hub A is "1". When hub B receives the subscription, it adds a

subscription object 1104 connected to a neighbor object for hub C, indicating that the subscriber can be reached through hub C.

Hub B then routes a subscription to a neighbor hub that is on a least-cost path to the publisher of the advertisement for the event. In the example, hub B will route the subscription to hub A, since the cost from hub A to itself is "0". When hub A receives the subscription, it creates a subscription object 1106 connected to a neighbor object for hub B, indicating that the subscriber can be reached through hub B. Hub A does not route the subscription further, since it is the originating hub for the advertisement that is being subscribed to.

Once the data structures of Fig. 12 have been established, an event published by publisher P will be sent hub-to-hub along the path defined by the subscription objects 1106, 1104, and 1102 until it reaches subscriber S.

Fig. 13 is a flow chart showing details of steps performed by the hubs of Fig. 1 to send published events to subscribers by routing the events through the hubs of the system. The steps of Fig. 13 are performed by 1) a hub that receives an event from its connected publisher (step 1202) or 2) a hub that receives an event from a neighboring hub (step 1203). The steps of Fig. 13 are performed by various hubs as each event is routed through the system. In the following paragraphs, the hub performing the steps of Fig. 13 is termed the "current hub."

If the current hub receives an event from one of its connected

publishers (step 1202), the preprocessor 514 of the current hub initially adds an "envelope" to the event (step 1204). A format of an event envelope is described below in connection with Fig. 9.. Preprocessor 514 then determines in step 1204 whether the event type of the event is registered in the hub. If not, the publisher cannot publish the event. If so, in step 1206, the hub places the event in the hub's incoming event queue for further processing. In the described embodiment, the event queue discards duplicate events (as determined by a sequence number or time stamp of the event envelope).

If the current hub receives the event from a neighboring hub (step 1203), the neighboring hub places the event in the incoming event queue of the current hub.

In step 1208, filter 518 of the current hub gets the event from the current hub's incoming event queue. Filter 518 then increments the time-spent field of the envelope, which indicates an amount of time spent in queue. In step 1210, filter 518 adds a routing block to the event. Each hub that routes the event adds another routing block to the event. A format of two routing blocks is shown in Fig. 10.

In step 1212, the current hub locates a "current" AdSource object that represents all the subscribers which are interested in the event coming from that particular AdSource. This will usually represent a least a cost path between the current hub and the subscribers, (note that the described embodiment does not reroute once an AdSource data structure

is established. The AdSource object is located by either searching all RemoteAds for the current hub (if the event came from a publisher attached to the current hub) or from searching the AdSourceRef list for the neighboring hub (if the event came from a neighboring hub). Steps 1216-1222 send the event to one or more subscribers. Steps 1216-1222 are performed for each sink object attached to the AdSource. Thus, steps 1216-1222 are performed for each subscriber that has asked to receive events of the type being routed.

If, in step 1216, the time-to-live for the event is less than the time-spent field of the envelope, then the event has expired and is not routed further by the current hub (step 1218). Otherwise, in step 1220, filter 518 determines whether the event has contents that fall within parameters specified by the subscriber. The subscriber specified the content filter at the time it subscribed to the event type (see field 774 of Fig. 8). For example, a subscriber may have requested to receive only those SalesEvents having where the customer lives in Los Angeles. If the event has values in the range specified by the subscriber, then the event is sent to the subscriber in step 1222 (either directly or by way of its hub, as described below). Each subscriber may have specified a different content filter (or no content filter). For example, a first subscriber may have specified that it wishes to receive SalesEvents where the customer lives in Los Angeles and a second subscriber may have specified that it wishes to receive SalesEvents where the customer lives in San Francisco.

In this example, it is possible that the event will match the first subscriber but not the second subscriber.

In step 1222, filter 518 sends the event to the matching subscriber. The subscriber may be a client of the current hub, in which case the event is sent directly to the subscriber. Alternately, the subscriber may be connected to a neighboring hub, either directly or indirectly. If a matching subscriber is connected to a neighboring hub, the current hub will route the event to the neighboring hub (unless the neighboring hub is the hub originating the event, as determined from field 802 of the envelope, see Fig. 9). The current hub also will not route the event to a hub that has already seen the event (as determined from the routing blocks attached to the event, see Fig.10).

In the described embodiment, the current hub only routes the event to a neighboring hub a single time, even if the neighboring hub has more than one matching subscription. Thus, if subscription object 651 of Fig. 6 matches the event, the event is forwarded to neighboring hub B. If subscription object 652 also matches the event, it is not necessary to forward the event to the neighboring hub B a second time. When neighboring hub B receives the event, it also will perform the steps of Fig. 13 to determine whether any of its client subscribers match the event.

Fig. 9 shows a format of an envelope data structure of an event. The preprocessor of Fig. 5 adds an envelope to an event received from its subscriber before the event is published. An envelope is associated

with each event and includes an origin hub 802, an adname 804, a publisher name 806, a territory name 808, an initial time stamp 810, a time to live 812, a priority 814, flags 816, and a time spent field 822. Flags 816 are unused in the described embodiment. In an alternate embodiment, the flags may indicate, e.g., whether an event is "persistent" or "transient."

Fig.10 shows an example format of two routing blocks of an event. Each hub adds a routing block to an event that it routes through itself. The API also includes a get_current_route_info routine that returns information about the actual route that an event took to reach a certain subscriber. Routing information is stored by the system as linked list, one entry for each hub visited. Each entry includes: an outgoing time stamp for the current hub, a hub name of the current hub, and a territory name of the current hub. The routing information optionally includes a sixty-four bit sequence number initially assigned to the event by the publisher.

V. Territories

Each territory 130 and 140 of Fig. 1 is a collection of hubs with a common event dictionary. Territories reflect organization, and are not necessarily tied to geography or direct physical connectivity. In the described embodiment, all hubs within a territory must be fully connected, because advertisement information is not forwarded across territories. Thus, there is always some path, although it may be indirect, between two

hubs within a territory without having to leave the territory. In a preferred embodiment, hierarchies of territories exist in which lower level territories inherit the event dictionaries of their parent territory. For example, a corporation may choose to have a territory for each subsidiary, with all of them understanding certain "corporate events."

A hub preferably may belong to only one territory. When a client (a publisher or a subscriber) connects to a hub, it specifies to which territory or territories it belongs. A client must belong to a territory because the meaning of an event, for example a SalesOrder, may be different in different territories. In the described embodiment, each hub has a predetermined default territory that is used if the application developer does not specify a territory.

In an alternate embodiment, multi-territorial hubs are especially useful for inter-company communication. Two companies that decide to communicate using the described embodiment of the present invention may set up a joint territory, agreeing upon some event types and a format to be shared. The hubs connecting the two companies will only forward events of this joint territory, since the territory information is kept as part of the advertisement, as described below in connection with event routing.

VI. Database Connectivity

Fig. 14 is a block diagram of a data base application 1302 incorporated into the network 120 of Fig. 1. Database application 1302,

which includes a database library having interface routines 1304, allows users to access information in data base 1308. Data base 1308 can be a data base manufactured by, for example, Oracle, Sybase, or Informix. The invention can be implemented using any appropriate data base that is capable of operating in conjunction with a transaction monitor.

Fig. 14 also includes a transaction monitor 1306. Transaction monitor 1306 preferably is the Encina transaction monitor, manufactured by Transarc Corporation. It can also be, for example, the Tuxedo transaction monitor manufactured by Novell. Alternately, any appropriate transaction monitor can be used to implement the present invention. A data base connecter 1310 having a data base library 1304 connects to network 120 to transmit and receive data from network 120. Data base connecter 1310 may include software performing hub functions or may connect to a separate hub. In Fig. 14, data base application 1302 sends information to data base 1308. Data base application 1302 also requests information from data base 1308 and displays that information in human readable form.

Data base connector 1310 can be a publisher, a subscriber, or both. If data base connector 1310 is a publisher, for example, it may publish an event each time a value is changed in data base 1302. It could also, for example, publish events at regular time intervals, reflecting the data that has been updated in the data base since the last publication. If data base 1302 is a subscriber, it stores the information of events it

receives in data base 1302. The described embodiment includes a configuration file (not shown) for data base connecter 1310. The configuration file specifies, for example, whether the database connection will be a publisher, a subscriber, or both. The configuration file also includes, for example, a description of events to publish and/or subscribe to. The configuration file also specifies information concerning the layout of data base 1308.

Fig. 15 is a flow chart 1400 showing steps performed when the data base is a subscriber. For example, as events are received from network 120, data from the events are added to data base 1302. Flow chart 1400 assumes that data base connector 1310 has already registered itself and its subscription(s), as described above. When an event is received by data base connector 1310 in step 1402, the data base connector maps and stores the data in the event into data base 1306. The data is not "committed" to the data base until the data has been completely stored in the data base. "Commitment" means that the data is actually entered in the data base. Until the changes to the data is committed, they are kept in a temporary storage location. Thus, uncommitted data base can be "rolled back" and the data transaction cancelled at any time up until the "commit" operation is performed. During a "roll back" the changes in the data in the temporary storage are discarded and the data base is retained in its state before the changes were made. Under certain circumstances data base connector 1310 will

not issue a "commit" command until a certain event is received.

Fig. 16 is a flow chart 1500 showing steps performed when the data base is a publisher. For example, as a human user adds, deletes and modifies data in the data base 1302, data connector 1310 sends periodic events to network 120. In step 1502, the transaction monitor 1306 is informed that data base connector 1310 is a "node." The exact manner in which this step is performed will vary, depending on the manufacturer of the transaction monitor 1306, but the step generally is well-known to persons of ordinary skill in that art. In step 1504, the data base connector 1310 registers itself and its advertisement(s), as described above.

Steps 1506-1512 show steps performed, e.g., as the user makes changes to data base 1308. When a data base transaction begins in step 1506, data base 1308 informs transaction monitor that a transaction has begun, using, e.g., X/Open's XA protocol. Transaction monitor 1306 informs data base connector 1310 that a transaction is occurring, and data base connector 1310 notes the transaction. In step 1508, changes are made to the data base by the user, but the user has not committed the changes).

If, in step 1507, the user (or software making changes) indicates that the transaction should commit, this fact is communicated to transaction monitor 1306 in step 1510. Transaction monitor 1306 informs data base connector 1310, which pulls the committed data from data base

1308 and publishes an event including the committed data. Events are published as shown in Fig. 13. If, in step 1512 the user (or software making changes) indicates that the transaction should roll back, this fact is communicated to transaction monitor 1306. Transaction monitor 1306 informs data base connector 1310, which "forgets" about the changes of step 1506 and does not publish an event. Thus, the described embodiment takes into account the commitment and roll back operations common to commercial data base operation.

In an alternate embodiment, where the data base product 1302 used does not include a transaction monitor, data base connector 1310 can act as a transaction monitor when connecting with data base 1308. Data base connector 1310 still acts as a publisher and/or subscriber in this configuration. Data base 1308 thinks that it is connected to a transaction monitor, when it is really connected to data base connector 1310.

VII. Summary

In summary, a preferred embodiment of the present invention includes a plurality of "publisher" entities, who publish information, and a plurality of "subscriber" entities, who request and use the information. Publishers and subscribers are connected to each other through a network. The network is a "store and forward" network whose routing is "content-based."

In the described embodiment of the present invention, the basic quanta of information is called an "event." Publishers publish events and subscribers subscribe to events that match criteria defined by the subscriber. Publication and subscription are performed asynchronously. Publishers and subscribers do not have direct knowledge of each other. The system receives published event from a publisher and routes the event to all appropriate subscribers. Each subscriber is guaranteed to receive all events published on the system if, and only if, they match the subscription criteria specified by the subscriber.

The described embodiment of the present invention makes it easy to integrate legacy systems, legacy applications, and legacy hardware into the system and attempts to minimize the amount of information that a user must learn to use the system. A legacy data base can be added to the network by way of a data base connector, which can be a publisher, a subscriber, or both.

Other embodiments will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope of the invention being indicated by the following claims.

APPENDIX A

6.0 The System Events

Twenty-three system events are used to communicate between hubs. They share some of the header structure of normal events, but are otherwise totally different and processed outside of the normal event code path.

The system event structure is defined in include/nxs_syspackP.hh:

```
typedef struct nxsSystemEventHeader {
        nxsULong length;              Byte length of the event
        nxsULong magic_number;     Used to detect bogus events
        nxsULong nexus_version;          Nexus version number
        nxsULong routing_offset;         nxsEventRoutingEntry
        nxsULong sys_event_type;              NXS_SYS_EVENT_*
        nxsULong data_offset;                    String data
} nxsSystemEventHeader;
```

The values for sys_event_type can be found in transport/hub/sysevent.hh. All system event processing code is declared in nexushub.hh (as methods of NexusHub) and defined in sysevent.cc.

The system event data is a string composed of spaced separated tokens. The possible tokens are:

string	string of any characters except <space> and <nul>
/string/	a string of any characters except ^_ (\037) surrounded by ^_ (\037)
long	a signed 32-bit value
ulong	an unsigned 32-bit value

Object references are transfered in stringified form.

System events are used for several styles of communication. During connection establishment, system events are used as peer-to-peer messages. These events are not seen by other hubs. Most other events affect the state of the territory and may be forwarded throughout the territory graph. A few events may be sent from a hub to itself.

NXS_SYS_EVENT_CONNECT_HUB

```
/string/    Sending hub name
/string/    Sending hub territory
string      Sending hub incoming EventQueue::PushConsumer
ulong       Receiving hub's cost
ulong       Sending hub's cost
```

This event is sent as a result of a call to NexusAdmin::HubAdmin::connectHub. The sending hub has already created the Neighbor and EventQueue for the connection. The receiving hub does likewise when it receives the system event. Note that the sending hub already has the incoming EventQueue::PushConsumer of the receiving via an argument to NexusAdmin::HubAdmin::connectHub. If this is the first connection for the receiving hub (it is joining a territory), then it needs to get the territory's shared data (event types, etc.). System events are used to request this information from the sending hub: NXS_SYS_EVENT_REQUEST_EVENTTYPES, and NXS_SYS_EVENT_REQUEST_ADVERTISEMENTS. If the hub is already a member of the territory, it sends its routes to the sender with NXS_SYS_EVENT_NEW_ROUTES. The receiving hub always requests routes from the sender with NXS_SYS_EVENT_REQUEST_ROUTES. If the connection cannot be created, for example the territory name is wrong, the receiver sends a NXS_SYS_EVENT_CONNECT_FAILED.

NXS_SYS_EVENT_CONNECT_FAILED

```
/string/    Sending hub name
/string/    Sending hub territory
/string/    Reason for connection failure
```

Sent when connection establishment failed. Clean up data structures.

NXS_SYS_EVENT_DISCONNECT_HUB

```
/string/    Sending hub name (or neighbor name)
/string/    Sending hub territory
```

This system event can be sent to self or to another hub. Sent to self as a result of call to NexusAdmin::HubAdmin::disconnectHub. In this case, the first argument is the name of the neighbor hub. The hub sends a similar event to the neighbor (containing the name of the sending hub) and does a connection shutdown. This may cause the following system events to be sent: NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS, NXS_SYS_EVENT_DELETE_ROUTES, and NXS_SYS_EVENT_DELETE_ADVERTISEMENTS. One the connection clean-up events are sent, a NXS_SYS_EVENT_END_OF_STREAM is delivered.

The sequence of events is the same on the neighbor hub receiving NXS_SYS_EVENT_DISCONNECT_HUB.

NXS_SYS_EVENT_END_OF_STREAM

```
/string/      Sending hub name
/string/      Sending hub territory
```

Indicates the last event to be seen on a connection. Clean up data structures.

NXS_SYS_EVENT_REQUEST_EVENTTYPES

```
/string/      Sending hub name
/string/      Sending hub territory
```

Send event types to the sender with NXS_SYS_EVENT_NEW_EVENTTYPE_FILES.

NXS_SYS_EVENT_NEW_EVENTTYPE_FILES

```
long          Number of files F
long          Number of types T
              Repeated F times:
/string/      .nexus file contents
/string/      .ifr file contents
              Repeated T times:
string        Event type name
```

Load the IFR and Nexus files into the IFR and created the given event types. The event is forwarded to all connected neighbors except the sender and neighbors who have already seen the event.

NXS_SYS_EVENT_FORWARD_EVENTTYPE_FILE

```
ulong         Address of EventTypeFile state
long          Number of event types T
              Repeated T times:
string        Event type name
```

Send the event type file and event types to all connected neighbors. Only sent to self when eventtypes are created or updated.

NXS_SYS_EVENT_REQUEST_ADVERTISEMENTS

```
/string/      Sending hub name
/string/      Sending hub territory
```

Deliver all advertisements to the sending hub with a NXS_SYS_EVENT_NEW_ADVERTISEMENTS.

NXS_SYS_EVENT_NEW_ADVERTISEMENTS

```
long          Number of advertisements A
```

```
                Repeated A times:
/string/        Advertisement name
/string/        Event type name
long            Priority
string          Storage mode; "p" or "t" for persistent or tran-
                sient
ulong           Time to live (seconds)
/string/        Originating hub name
/string/        Originating hub territory
```

Create RemoteAds for the listed advertisements. The originating hub is where the advertisement was created. The system event is forwarded to all connected neighbors excepted the sender and neighbors who have already seen the event.

NXS_SYS_EVENT_CONNECTION_READY

```
/string/        Sending hub name
/string/        Sending hub territory
```

The sending hub is ready to use the connection. If the receiver is also ready, it replies with a NXS_SYS_EVENT_CONNECTION_READY. If the connection is already established the system event is ignored.

NXS_SYS_EVENT_FORWARD_ADVERTISEMENT

```
string          Advertisement name
```

Send the named advertisement to all connected neighbors. Only sent to self when advertisements are created.

NXS_SYS_EVENT_CHANGE_ADVERTISEMENT

```
/string/        Advertisement name
/string/        Originating hub name
/string/        Originating hub territory
long            Priority
string          Storage mode; "p" or "t" for persistent or tran-
                sient
ulong           Time to live
```

Update the matching RemoteAd with the given values. The system event is forwarded to all connected neighbors except the sender and hubs that have already seen the event.

NXS_SYS_EVENT_DELETE_ADVERTISEMENTS

```
long            Number of Advertisements A
                Repeated A times:
/string/        Advertisement name
/string/        Originating hub name
/string/        Originating hub territory
```

Delete the associated RemoteAds and related data structures. Sent by the originating hub when an advertisement is deleted. The system event is forwarded to all connected neighbors except the sender and hubs that have already seen the event.

NXS_SYS_EVENT_REQUEST_ROUTES

```
/string/    Sending hub name
/string/    Sending hub territory
```

Deliver all routing information to sender with NXS_SYS_EVENT_NEW_ROUTES.

NXS_SYS_EVENT_NEW_ROUTES

```
long        Number of routes R
            Repeated R times:
/string/    Advertisement name
/string/    Originating hub name
/string/    Originating hub territory
long        Cost
```

Record the existence of a route from the sender to the given advertisements at the given cost. AdSource objects are created and if this is the first route to the advertisement, the local clients are checked for subscription matches. This may result in a NXS_SYS_EVENT_NEW_SUBSCRIPTIONS to the sender of NXS_SYS_EVENT_NEW_ROUTES. Each route is considered separately for forwarding. The route is not forwarded to the sender, the originating hub, or to a hub that has already seen the event. The route is only forwarded if it is the first route for the RemoteAd, or it is the second route and the destination hub is current route provider. The cost field of the forwarded event is incremented by the cost of the connection over which the event was received.

This system event may mark the end of connection setup. If the event came from a neighbor that is not yet fully connected, NXS_SYS_EVENT_CONNECTION_READY is sent to the neighbor.

NXS_SYS_EVENT_CHANGE_ROUTES

```
long        Number of route changes R
/string/    Originating hub name
/string/    Originating hub territory
            Repeated R times:
/string/    Advertisement name
long        Cost
```

Change the cost of a route. Find the matching RemoteAd and AdSource and update its cost value. The system event is forwarded to all connected neighbors except the sender and neighbor who have already seen the event.

NXS_SYS_EVENT_DELETE_ROUTES

```
long        Number of routes R
            Repeated R times:
```

```
/string/        Advertisement name
/string/        Originating hub name
/string/        Originating hub territory
```

Delete the routes (AdSource) for the given advertisements (RemoteAd). Any subscriptions which were fed by that route are failed. This may result in NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS sent to neighbors who have subscriptions registered for the advertisement. Forwarding of each deleted route is considered separately and only if the last route to the advertisement was deleted. The system event is not forwarded to the sender or the originating hub.

NXS_SYS_EVENT_CHANGE_CONNECTION

```
/string/        Sending hub name
/string/        Sending hub territory
long            Delta to sending hub's connection cost
```

Update their cost for the connection.

NXS_SYS_EVENT_NEW_SUBSCRIPTIONS

```
long            Number of subscriptions S
                Repeated S times:
/string/        Advertisement name
/string/        Advertisement hub name
/string/        Advertisement hub territory
/string/        Filter expression
ulong           Owner id in sender
ulong           Subscription id in sender
```

Register subscriptions from the sender against the given advertisements. Create a Subscription on the sending neighbor and tell the RemoteAd about it. The RemoteAd tells the current AdSource which creates a Sink for the Neighbor (if one does not yet exist). The Sink adds a SubscriptionRef for the Subscription.

If the advertisement is from another hub, forward a system event for the new subscription to the neighbor supplying the current route.

The owner and subscription ids are used to locate the subscription when it is canceled and to identify it when it fails.

If there is a failure during subscription registration, an NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS is returned to the sender.

NXS_SYS_EVENT_CANCEL_SUBSCRIPTIONS

```
long            Number of subscriptions S
                Repeated S times:
ulong           Owner id in sender
ulong           Subscription id in sender
```

Cancel the listed subscriptions. This event can be sent from self or a neighbor. Remove all references to the subscription from all current AdSources. This should delete all reference to the Subscription by Sink owned SubscriptionRefs. The cancel is forwarded to all neighbors which could possibly feed the subscription.

NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS

```
long        Number of failures S
            Repeated S times:
ulong       Owner id in receiver
ulong       Subscription id in receiver
/string/    Hub name where failure occurred
/string/    Territory where failure occurred
/string/    Failure reason
```

Remove reference to the subscriptions from AdSources. The subscription itself is not affected by the failure. If the subscription was from a neighbor, the failure is forwarded back to the originating hub.

NXS_SYS_EVENT_FORWARD_SUBSCRIPTION

```
ulong       Client id
ulong       Subscription id
```

Forward the new subscription to neighbors which can feed it. Tell all matching RemoteAds about the subscription. Each RemoteAd tells its current AdSource which creates a Sink for the Client (if one does not yet exist). The Sink adds a SubscriptionRef for the Subscription. Send a NXS_SYS_EVENT_NEW_SUBSCRIPTIONS to the current source for each non-local RemoteAd.

NXS_SYS_EVENT_DELETE_EVENTTYPE

```
string      Event type name
```

Destroy the event type. The system event is forwarded to all connected neighbors except the sender and hubs that have already seen the event.

4. Brief Description of Drawings

Fig. 1 is a block diagram of a networked computer system in accordance with a preferred embodiment of the present invention.

Fig. 2 is a flow chart showing steps performed to install an application, such as a publisher, on a hub of Fig. 1.

Fig. 3 is a flow chart showing steps performed by a publisher of Fig. 1 to publish events.

Fig. 4 is a flow chart showing steps performed by a subscriber of Fig. 1 to subscribe to events.

Fig. 5 is a block diagram showing details of a hub of Fig. 1.

Fig. 6 is a diagram showing data structures stored in a memory of the hub of Fig. 5.

Fig. 7 is a listing of details of the data structures of Fig. 6 .

Fig. 8 is a diagram of additional data structures stored in the memory of the hub of Fig. 5 for the purpose of storing information about clients of the hub.

Fig. 9 shows a format of an envelope data structure of an event.

Fig. 10 shows a format of a routing block of an event.

Fig. 11 is a flow chart showing details of steps performed by the hubs of Fig. 1 to populate the data structures of Fig. 6 .

Fig. 12 is a diagram showing examples of the data structures of

Fig. 6 populated with data.

Fig. 13 is a flow chart showing details of steps performed by the hubs of Fig. 1 to send published events to subscribers.

Fig. 14 is a block diagram of a data base application incorporated into the network of Fig. 1.

Fig. 15 is a flow chart showing steps performed when the data base is a subscriber.

Fig. 16 is a flow chart showing steps performed when the data base is a publisher

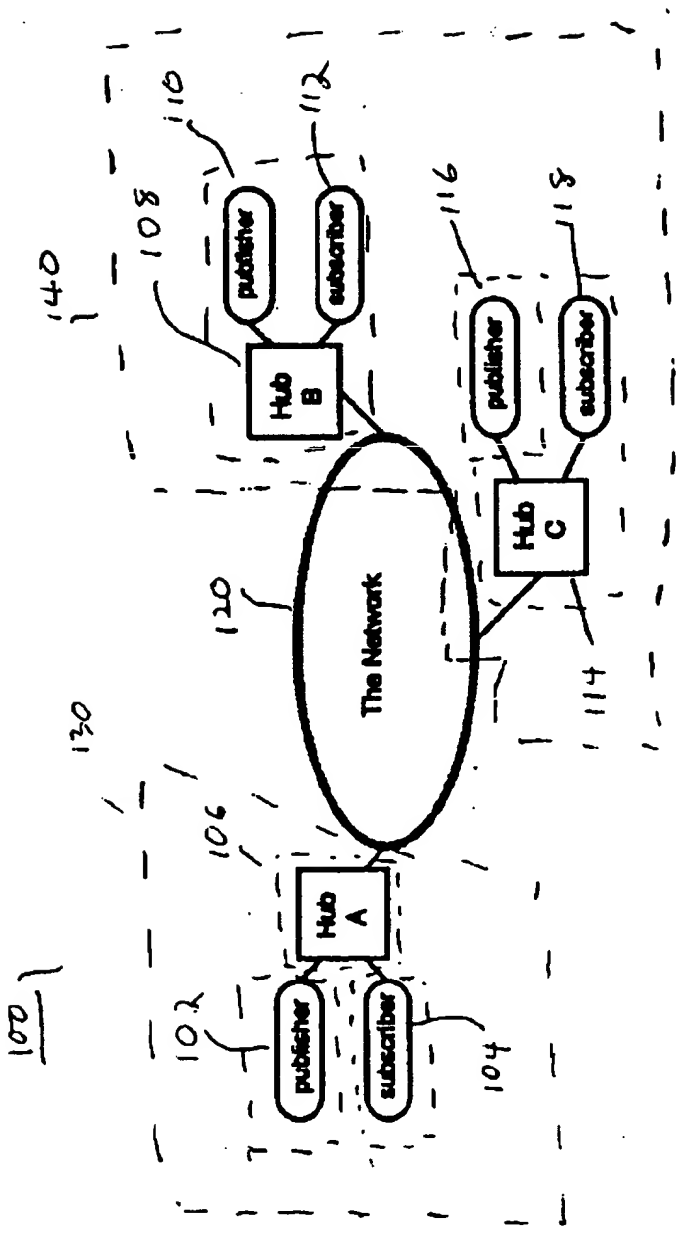

100
102
104
106
108
110
112
114
116
118
120
130
140
publisher
subscriber
Hub A
Hub B
Hub C
The Network

Fig. 1

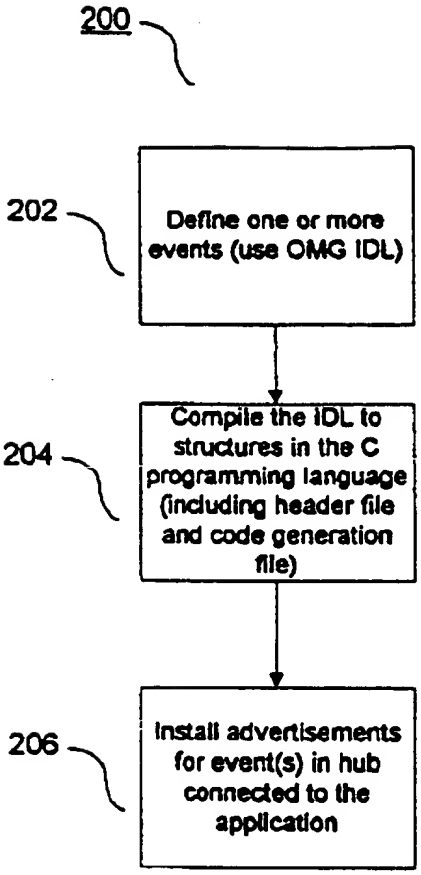
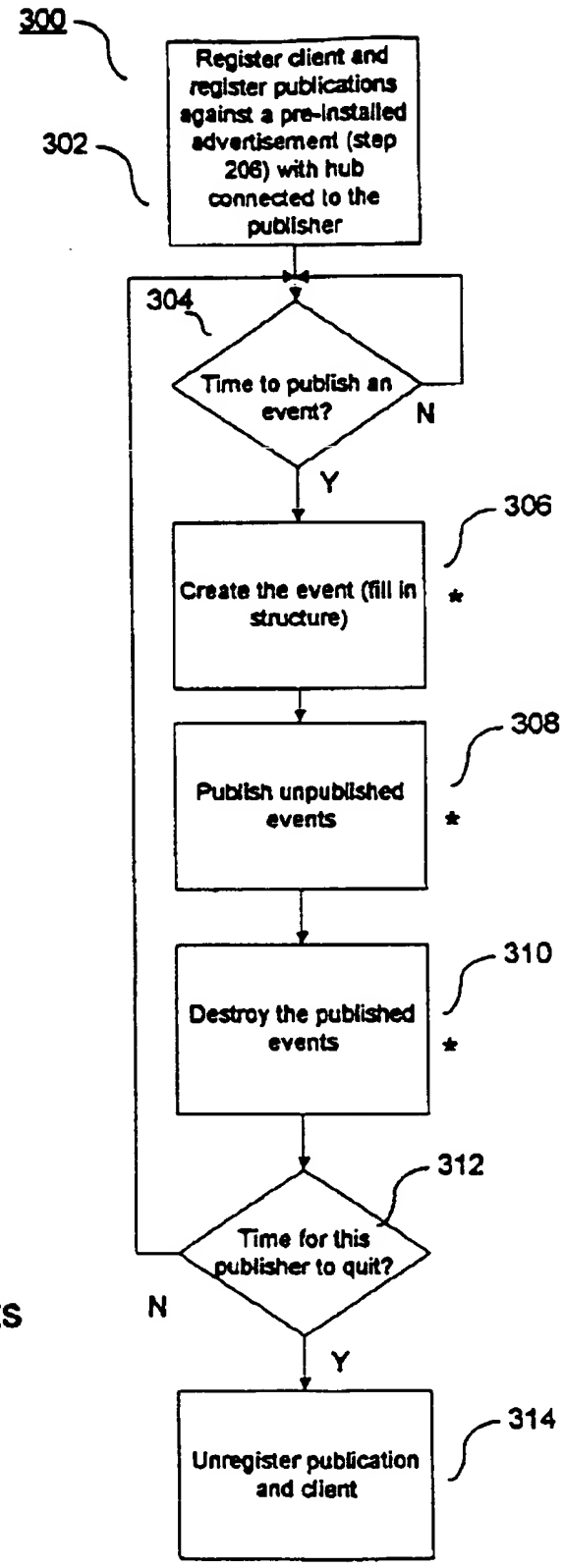
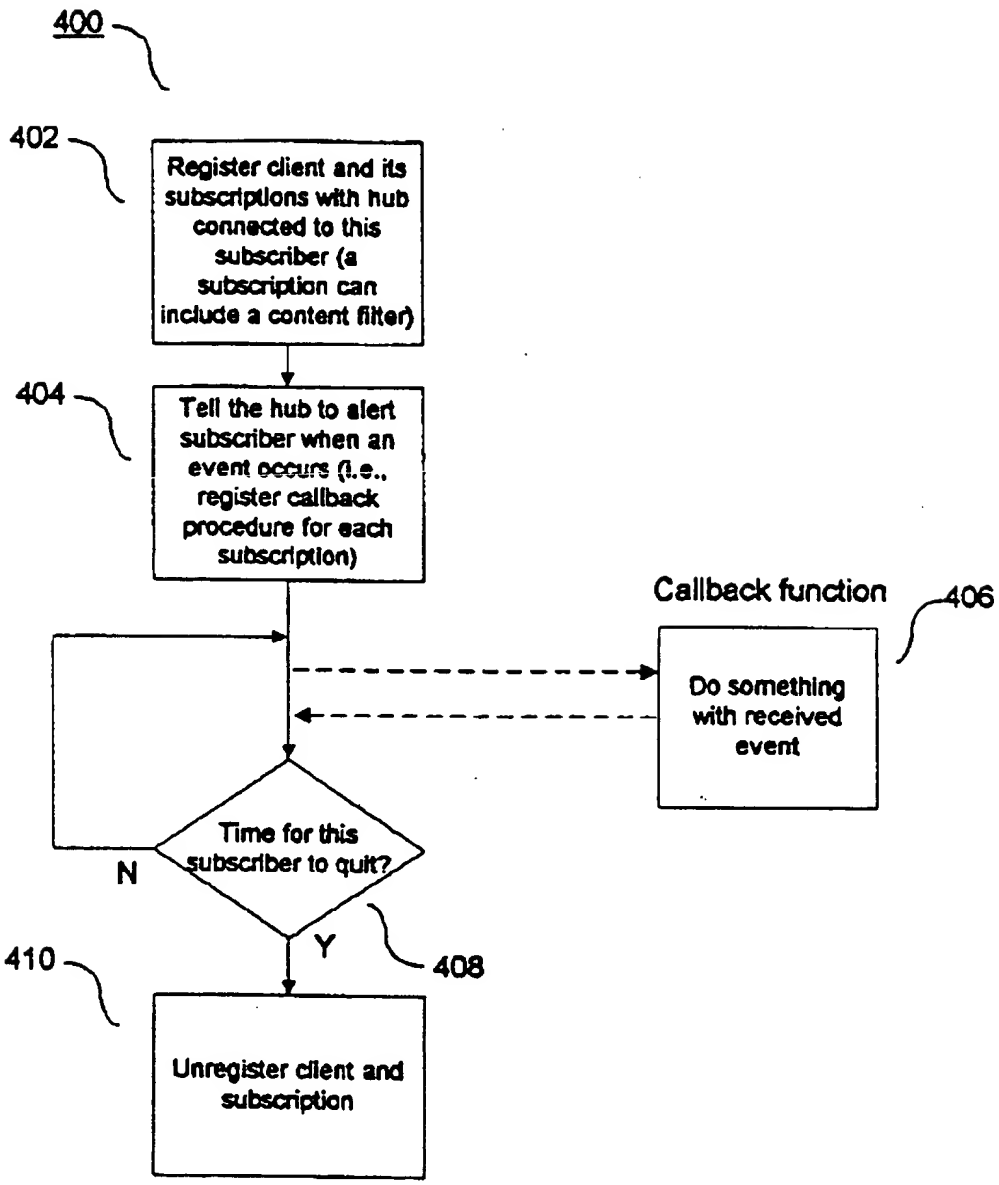


Fig. 2
Installing an Application (Publisher) on a Hub



Publishing Events
Fig. 3



Subscribing to Events

Fig. 4

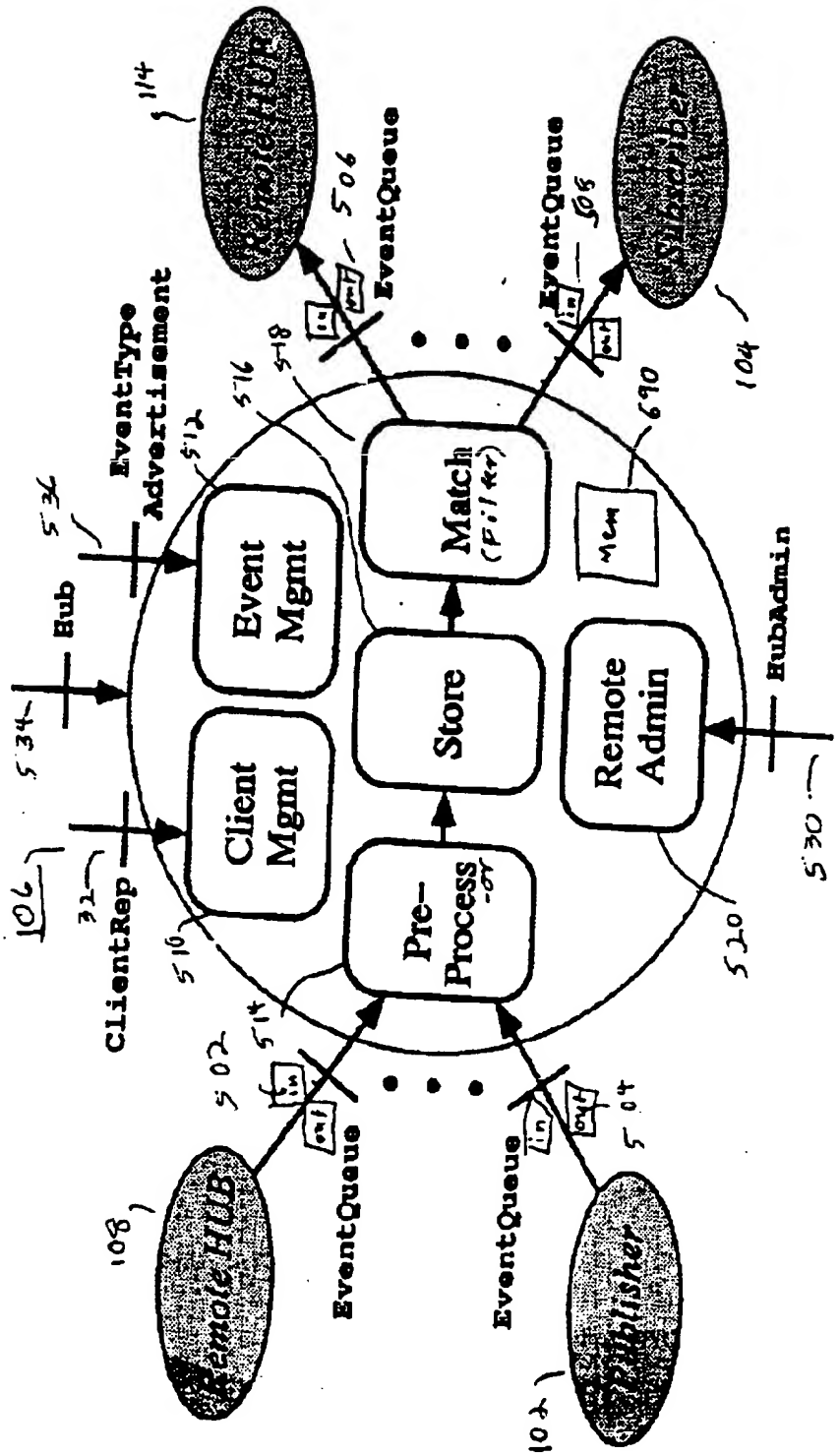

Remote HUB
Publisher
Subscriber
EventQueue
EventQueue
EventQueue
EventQueue
ClientRep
Hub
EventType Advertisement
HubAdmin
Client Mgmt
Event Mgmt
Pre-Process
Store
Match
Remote Admin
104
106
108
114
502
504
506
510
512
514
516
518
520
530
534
536
690

Fig. 5

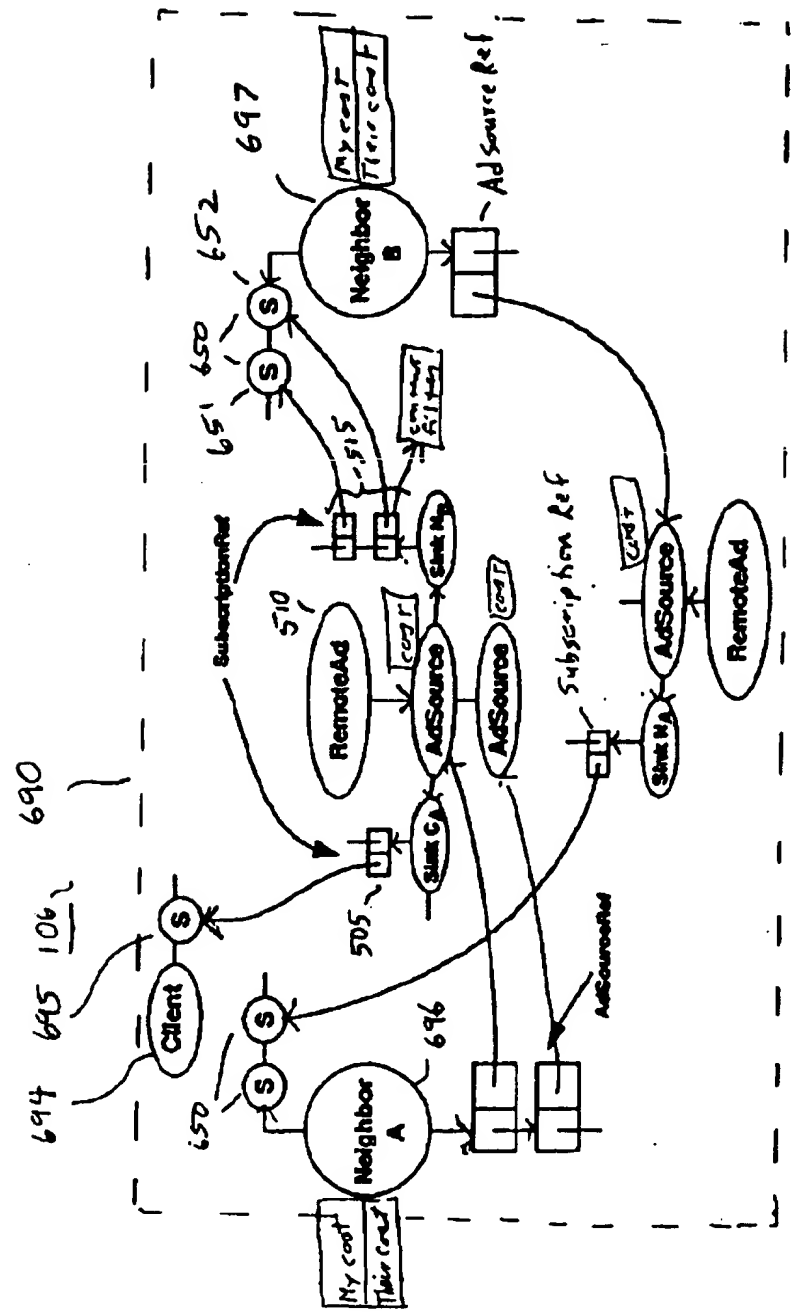

690
697
652
650
651
515
510
505
696
694
695
106
Neighbor B
Neighbor A
Client
AdSource Ref
Subscription Ref
RemoteAd
AdSource
Sink
S

Fig. 6

The following gives the important data fields of the classes in the above diagram.

Client: name, id, event queue, event queue pushconsumer, Subscription list, publication list

Neighbor: name, id, event queue, event queue pushconsumer, AdSourceRef list, Subscription list, my cost, their cost

AdSourceRef: pointer to AdSource

RemoteAd: name, eventtype name, priority, storage mode, time to live, originating hub, originating territory, AdSource list, current AdSource

AdSource: Sink list, cost

Sink: SubscriptionRef list

SubscriptionRef: pointer to Subscription

Subscription: eventtype name, filter expression, owner id, subscription id, reference count, (for remote subscriptions: neighbor owner id, neighbor subscription id, ad name, ad originating hub)

Fig 7

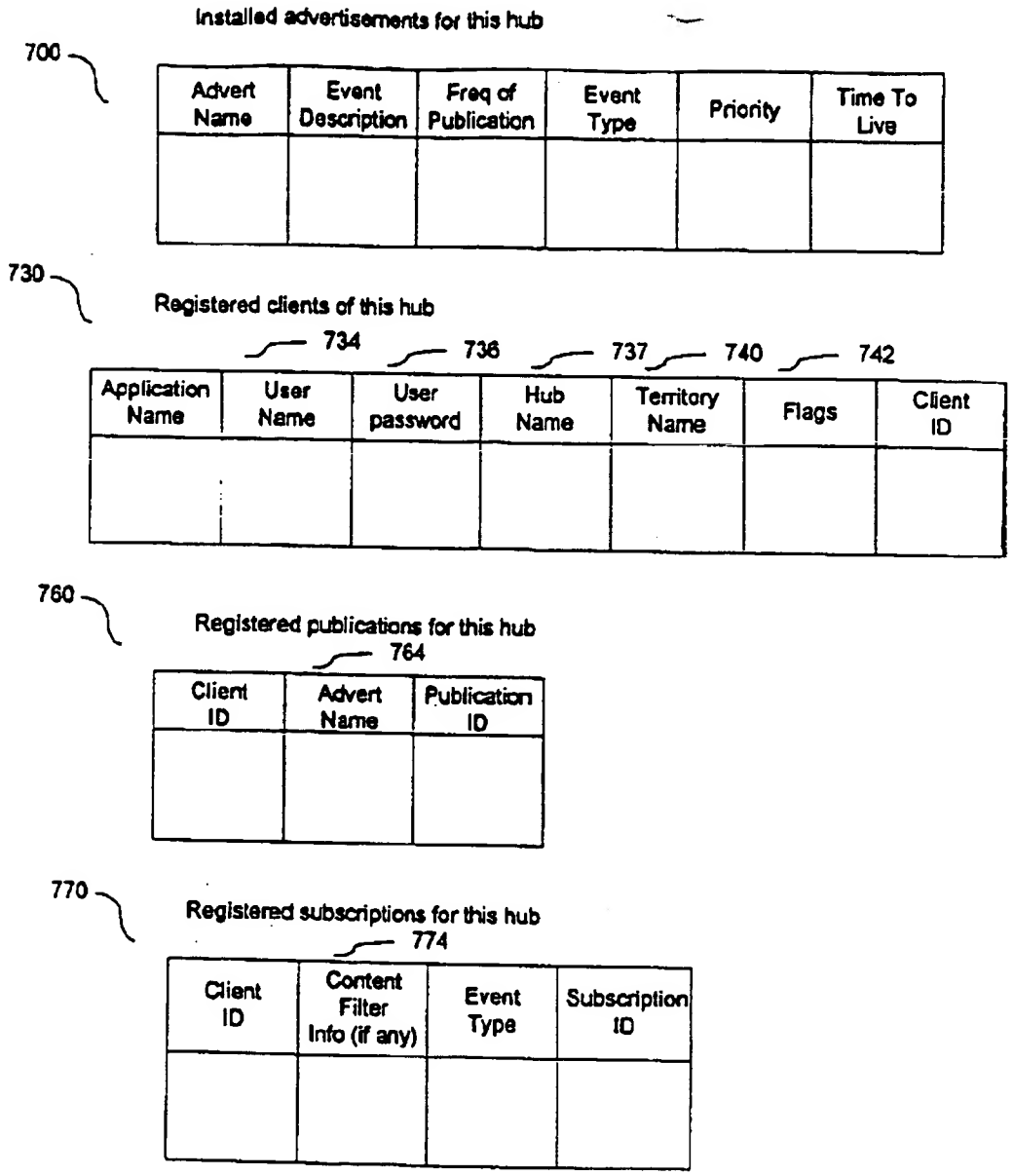

Installed advertisements for this hub
700
Advert Name
Event Description
Freq of Publication
Event Type
Priority
Time To Live
730
Registered clients of this hub
734
736
737
740
742
Application Name
User Name
User password
Hub Name
Territory Name
Flags
Client ID
760
Registered publications for this hub
764
Client ID
Advert Name
Publication ID
770
Registered subscriptions for this hub
774
Client ID
Content Filter Info (if any)
Event Type
Subscription ID

Fig. 8

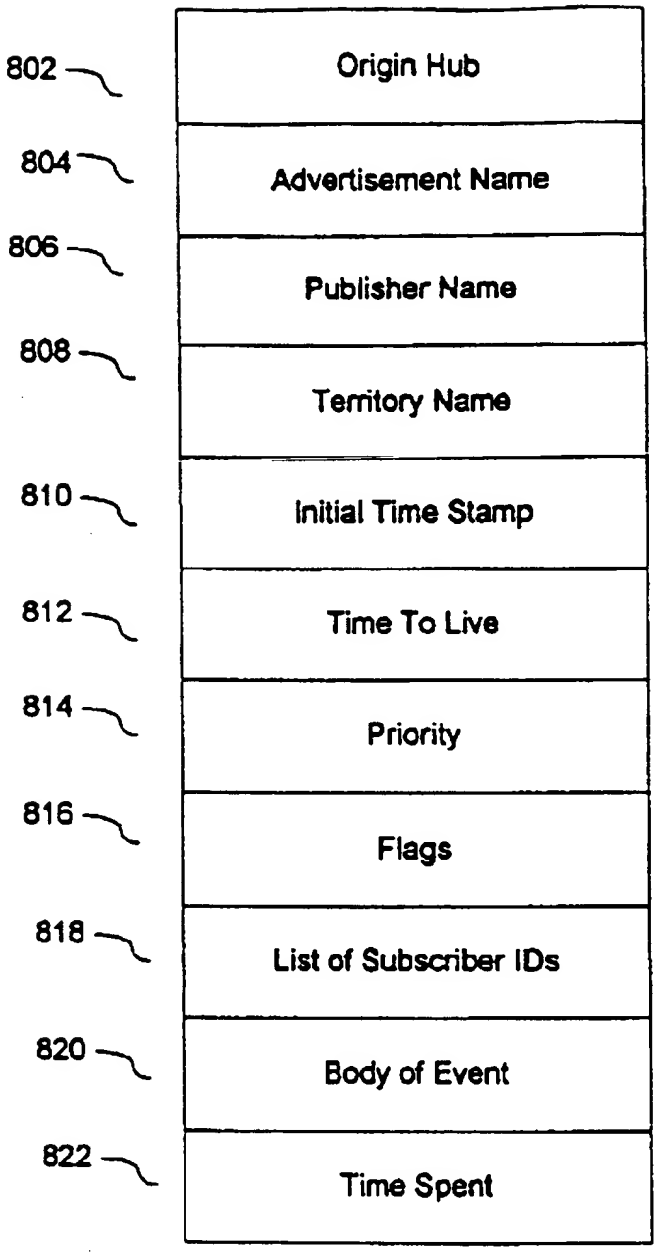


Fig. 9

Envelope Format of Event

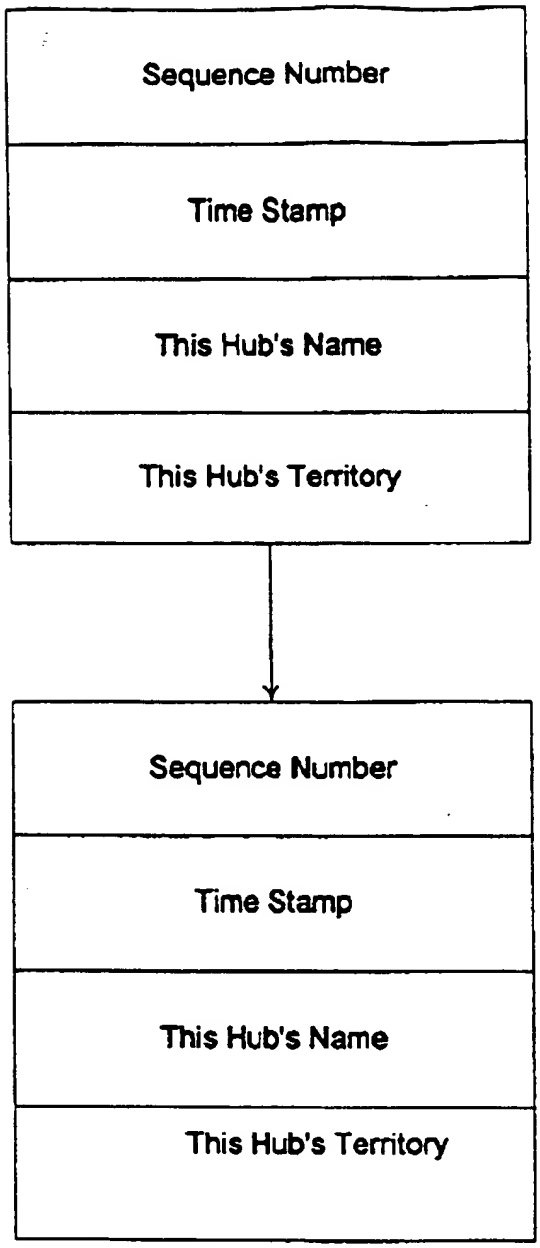


Fig. 10
Routing Blocks of an Event

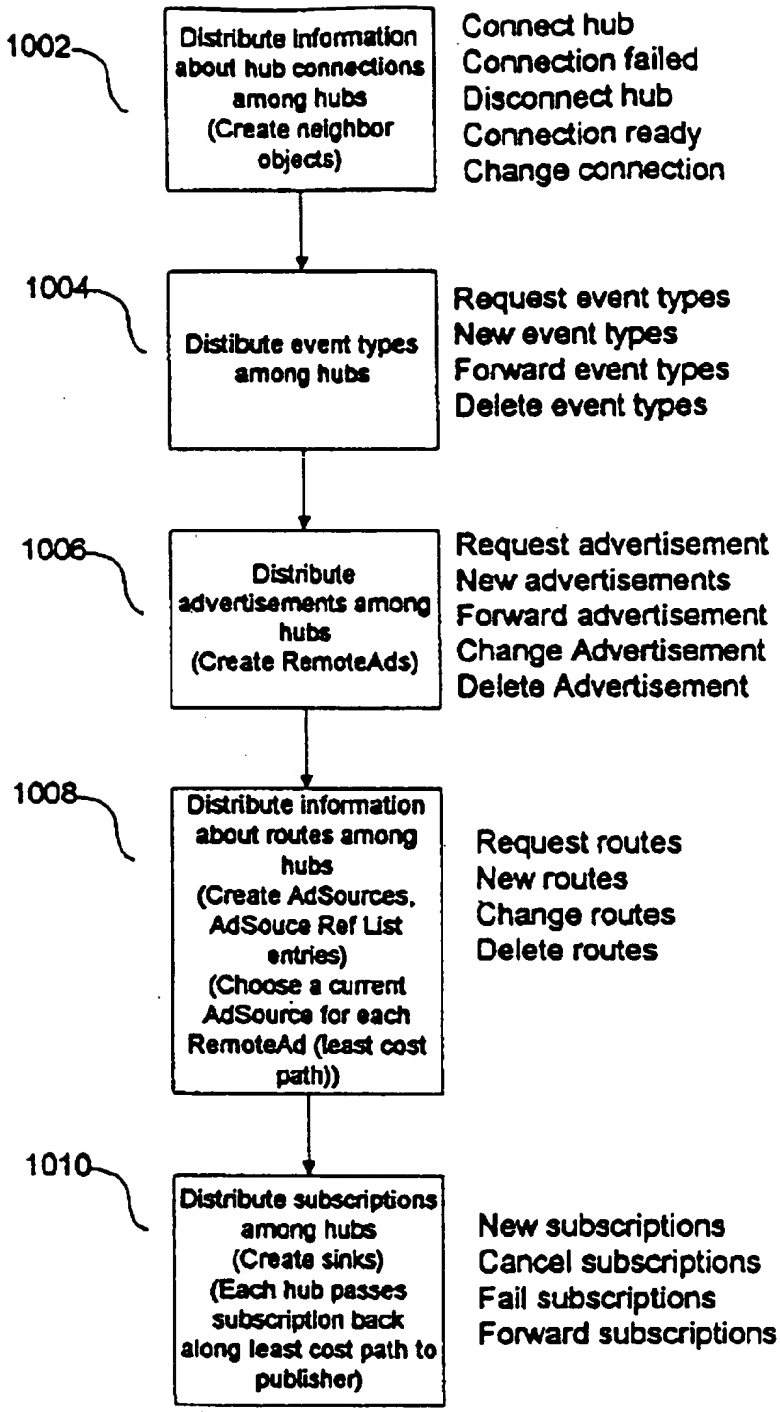


Fig. 11
Populating Data Structures of Hubs

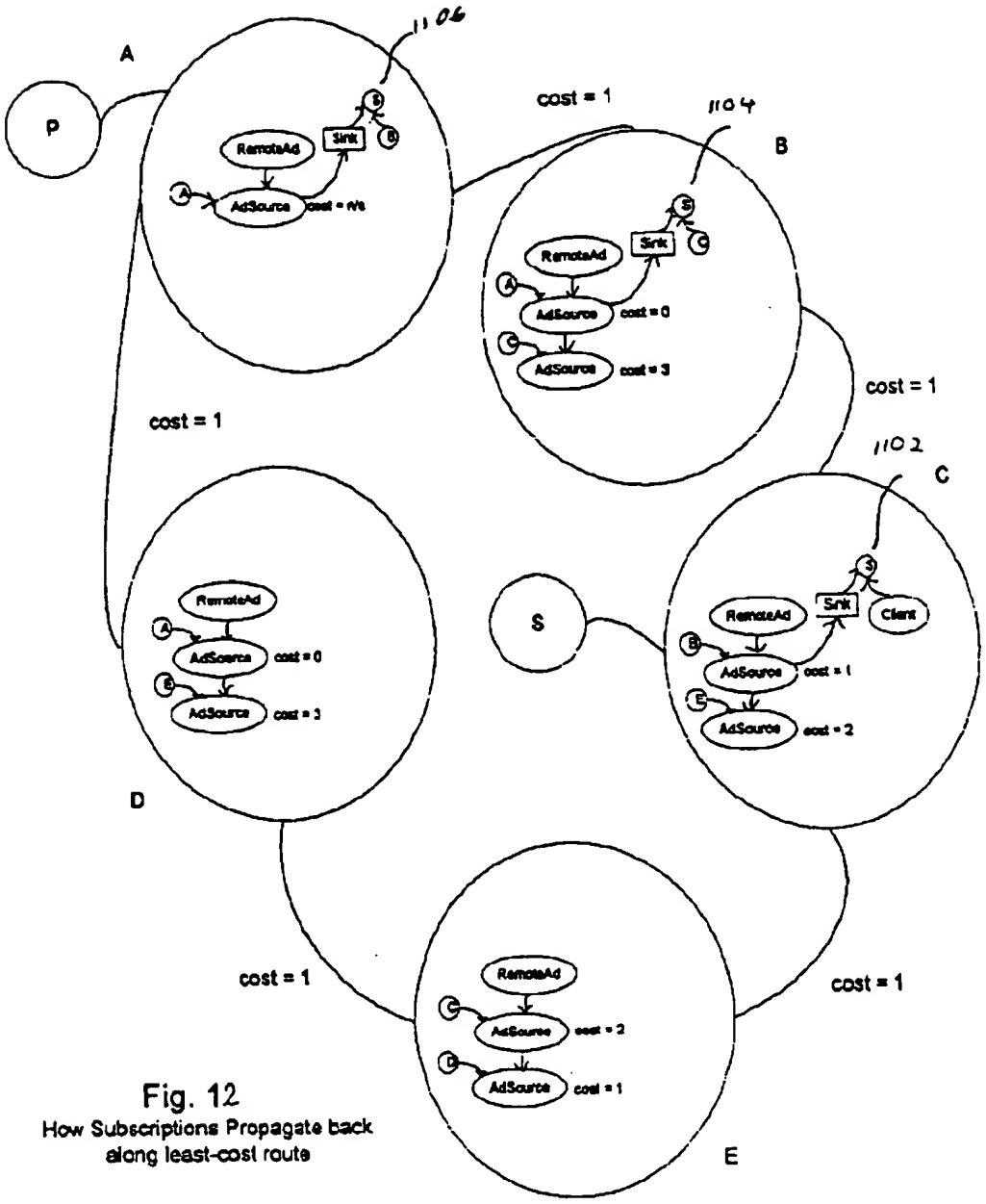


Fig. 12
How Subscriptions Propagate back along least-cost route

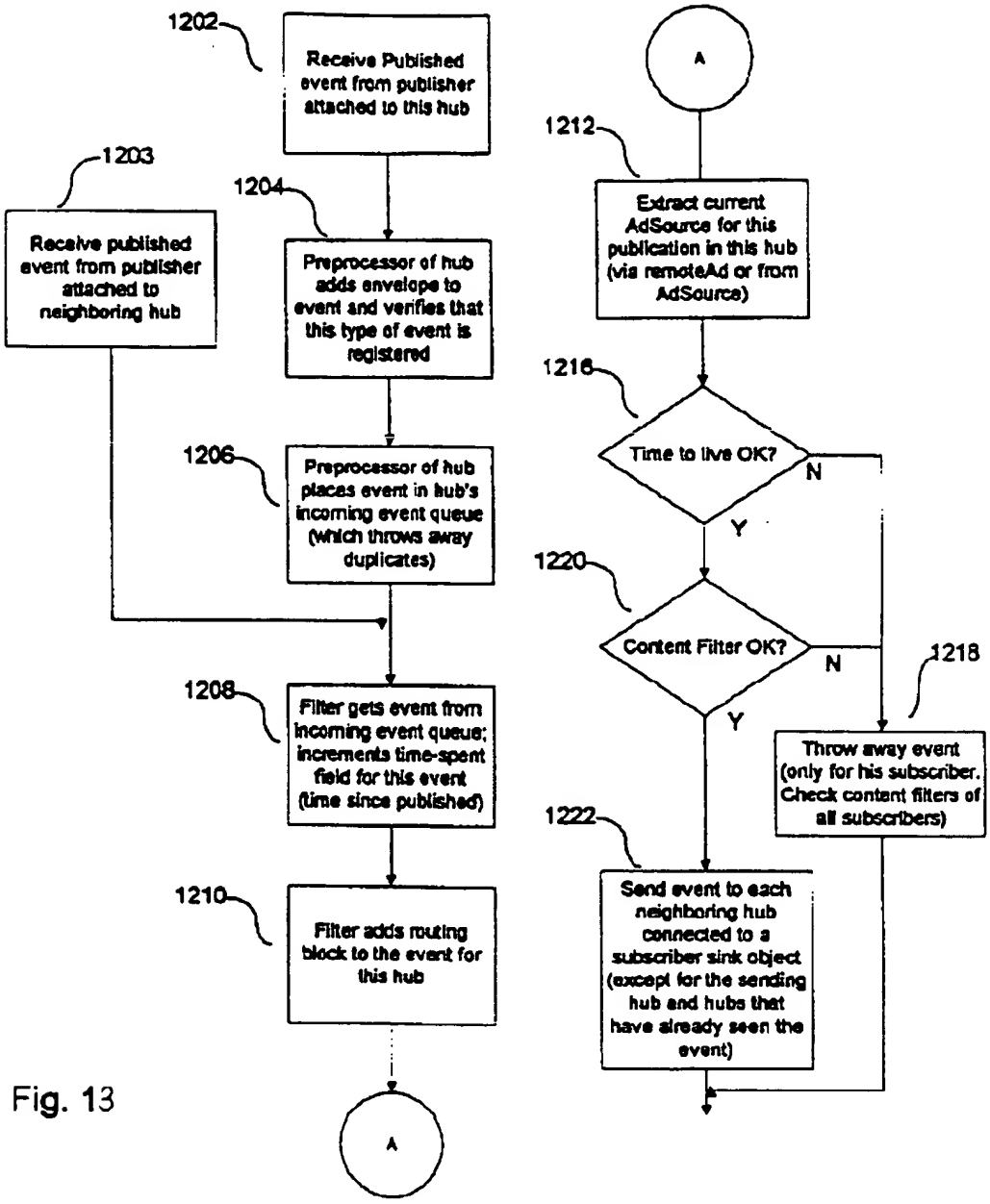

1202
Receive Published event from publisher attached to this hub
1203
Receive published event from publisher attached to neighboring hub
1204
Preprocessor of hub adds envelope to event and verifies that this type of event is registered
1206
Preprocessor of hub places event in hub's incoming event queue (which throws away duplicates)
1208
Filter gets event from incoming event queue; increments time-spent field for this event (time since published)
1210
Filter adds routing block to the event for this hub
A
A
1212
Extract current AdSource for this publication in this hub (via remoteAd or from AdSource)
1216
Time to live OK?
N
Y
1220
Content Filter OK?
N
Y
1218
Throw away event (only for his subscriber. Check content filters of all subscribers)
1222
Send event to each neighboring hub connected to a subscriber sink object (except for the sending hub and hubs that have already seen the event)

Fig. 13

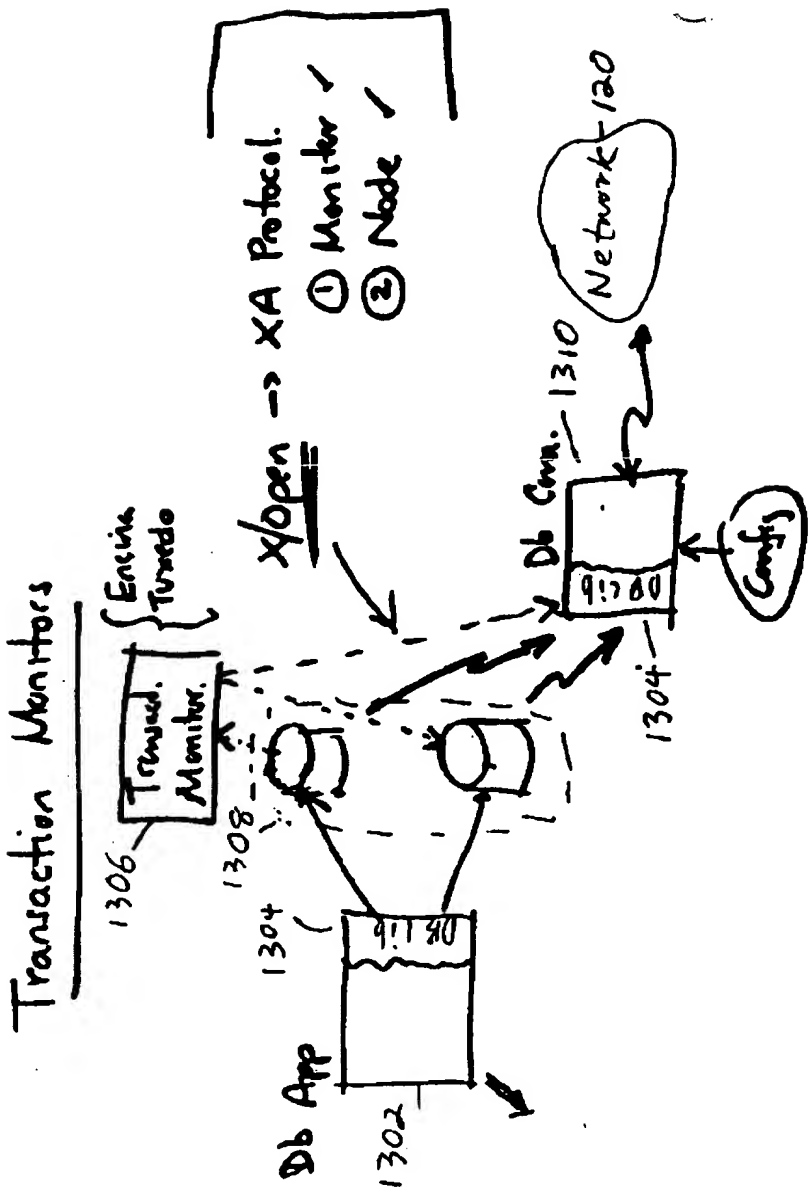

Transaction Monitors
Encina Toolkit
Transact. Monitor.
1306
1308
X/Open → XA Protocol
① Monitor
② Node
Db Comm.
1310
Network
120
Config
1304
1304
Db App
1302

Fig. 14

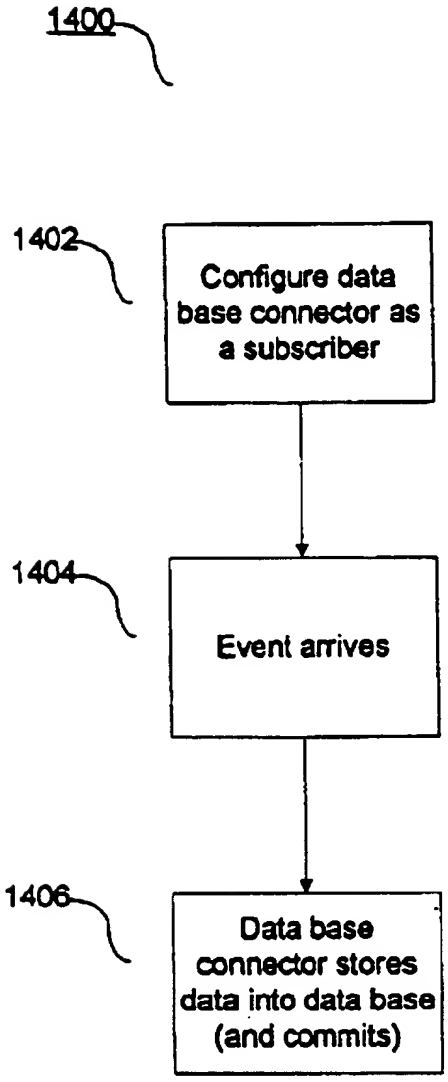

1400
1402
Configure data base connector as a subscriber
1404
Event arrives
1406
Data base connector stores data into data base (and commits)

Fig. 15

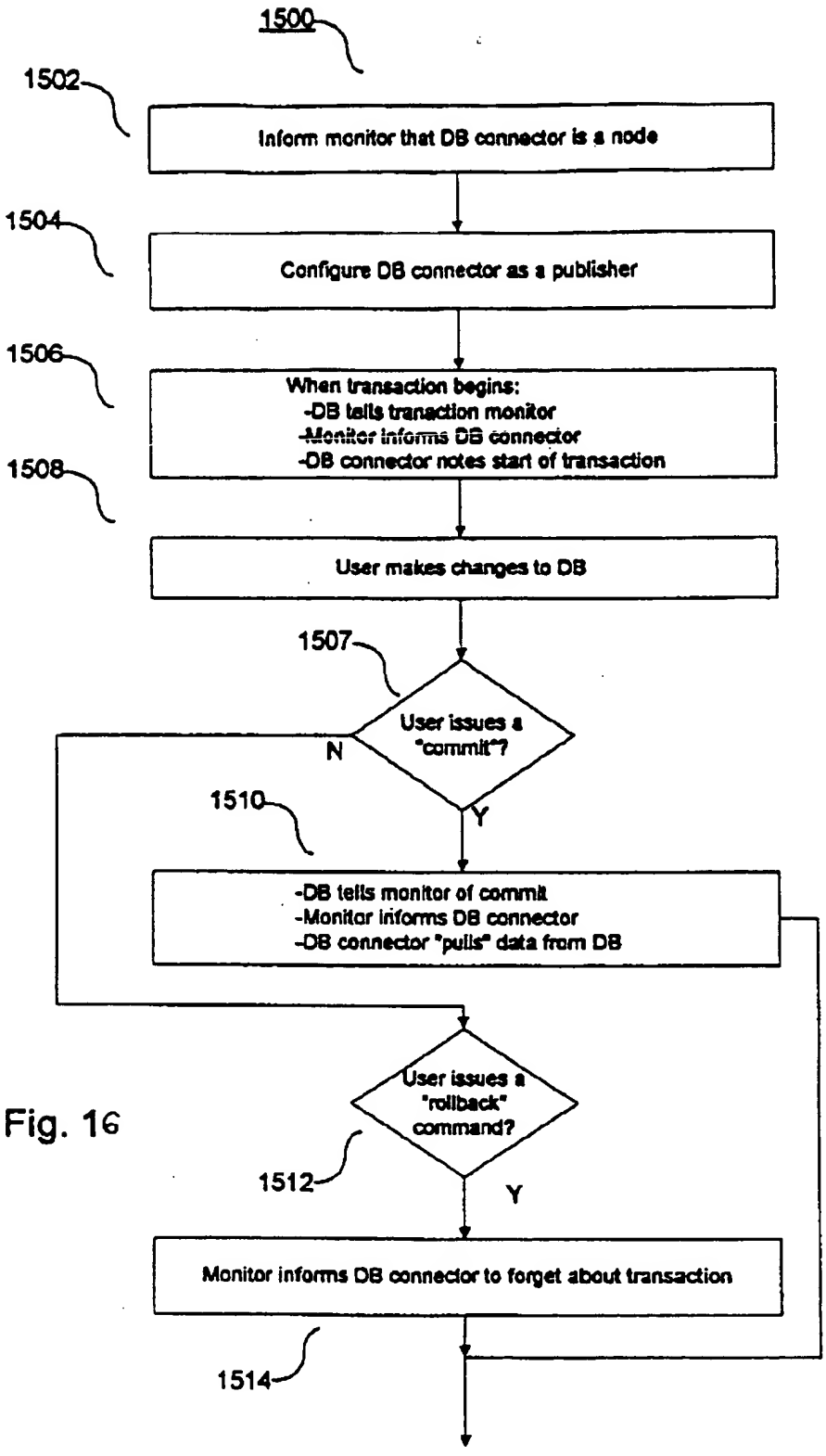

1500
1502
Inform monitor that DB connector is a node
1504
Configure DB connector as a publisher
1506
When transaction begins:
-DB tells tranaction monitor
-Monitor informs DB connector
-DB connector notes start of transaction
1508
User makes changes to DB
1507
User issues a "commit"?
N
Y
1510
-DB tells monitor of commit
-Monitor informs DB connector
-DB connector "pulls" data from DB
User issues a "rollback" command?
1512
Y
Monitor informs DB connector to forget about transaction
1514

Fig. 16

1. Abstract

A method and apparatus for publishing and receiving events to a network. A plurality of "publisher" entities publish information and a plurality of "subscriber" entities request and use the information. Publishers and subscribers are connected to each other through a network. The network is a "store and forward" network whose routing is "content-based." The basic quanta of information is called an "event." Publishers publish events and subscribers subscribe to events that match criteria defined by the subscriber. Publication and subscription are performed asynchronously. Publishers and subscribers do not have direct knowledge of each other. The system receives a published event from a publisher and routes the event to all appropriate subscribers. Each subscriber is guaranteed to receive all events published on the system if, and only if, they match the subscription criteria specified by the subscriber. A legacy data base can be added to the network by way of a data base connector, which can be a publisher, a subscriber, or both.

2. Representative Drawing

Fig.1